Atari Floppy Disk Copy Protection

By Jean Louis-Guérin (DrCoolZic)
Revision 1.3a – November 2014

Table of Contents

Chapter 1. Presentation

This document describes ***floppy disk protection mechanisms*** used on the Atari platform. This type of **copy protection** is very old and, with many years of development and the usage of sophisticated floppy disk hardware, it has conducted to numerous protection methods frequently referred as **key disk protection**. The key disk protection method has at least two obvious qualities: first, a *key disk* can be simultaneously used as *protection* and *distribution* disk and second, this type of protection is very cheap but nevertheless hard to tamper with. So, key disks have been widely used to protect Atari programs/games. To understand the key disk based protections, one is assumed to have some basic knowledge about FD/FDC data and operation.

Some of the FD protection mechanisms are generic to many platforms while some are directly related to a specific Floppy Disk Controller used on a specific platform. Therefore, in order to get a general understanding, I have reviewed the FD protections mechanism used on several platforms: Amiga, Commodore C64, PC, Tandy, Atari 8 bits and Atari ST 16 bits (see the references section).

A lot of information about the different copy protection mechanisms presented here has been collected from the Web. Links to the original information / Web sites can be found at the end of this document in the references section.

In order to validate this document, I have analyzed the protections of many original floppy disks. For that matter I have developed several programs over time:

- For detailed analysis of *timing information*, the first program that I have created is called ***Analyze.*** It runs on Atari and PC. This program reads files produced on Atari by the ***Discovery Cartridge*** and performs a detailed analysis of the flux reversals read from a diskette. This program takes its root in experiments I have done back in the 80s! The program is now replaced by the ***AUFIT*** program presented below.
- For basic protection analysis I have created a program running on **Atari** called ***Panzer*** (**P**rotection **AN**aly**ZER**) that automatically detects and reports most of the protections. This program also provides the capability to analyze and report detailed sectors and tracks information (including **timing** for track and sector).
- ***KFAnalyze*** program reads input **Stream** files generated by the KryoFlux board. As a Stream file provides Atari FD information at the flux reversals level (more information in the references section), it is possible to provide much more accurate detections of protections especially those related to bit cell timing variation. The heart of this program is a Western Digital WD1772 Floppy Disk Controller emulation. This emulation implements a full DPLL data separator and provides functions equivalent to the ***read track***, ***read address***, and ***read sector*** commands directly from the *Stream* files. Therefore it is possible to process the Stream information as if we were read by an Atari WD1772 FDC but with a lot of extra information especially on timing.
- ***KFPanzer*** (**K**ryo**F**lux **P**rotection **AN**aly**ZER**) is a program running on **PC** that analyzes the protection using information from **Stream** files generated by a KryoFlux board. It is a combination of the **Panzer** and **KFAnalyze** programs.
- My latest program for analyzing protection is called ***AUFIT*** (Atari Universal FD Image Tool). It combines the features of the above programs with a nice Graphical User Interface. This programs can read information at different levels (down to flux transition level: Kryoflux, Supercard Pro), analyzes and displays information about the protections, and can write different images formats for emulation.

I want to thanks to many people on Atari forum for taking time to discuss some of the protections presented here (See HERE and HERE).

Chapter 2. Copy protections detail description

In this section I provide a detailed description of the different protection's mechanisms used in Atari Key disks. The protections have been grouped into two categories:

- Protections based on data
- Protections based on timing

2.1 Protections based on data

This category contains protections based on using nonstandard data content in the tracks and/or sectors of a diskette.

A "normal diskette" has one or two sides (i.e. single or double sided) each having 80 tracks numbered from 0 to 79. A more detailed description of formats can be found in the Atari Low-Level Formats section.

A "standard track" on an Atari is composed of 9 sectors each with 512 bytes of data sequentially numbered from sector 1 until sector 9.

However it is not uncommon to use diskettes with up to 11 sectors and more than 80 tracks as it allows packing more data. A good duplication/imaging program should be able to detect and reproduce all these alternatives and therefore they are not really considered as protection.

But beyond these basic variations of a diskette's data content we will see that some protections uses mechanism **difficult to detect** (so that a copy program would not easily find them) and some that **cannot be reproduced** without special hardware.

2.1.1 Number of tracks (NOT)

A “normal Atari diskette” has 80 tracks numbered 0 through 79 on each side. Some protections are based on extra or missing tracks.

2.1.1.1 Extra tracks (NOT-EXT)

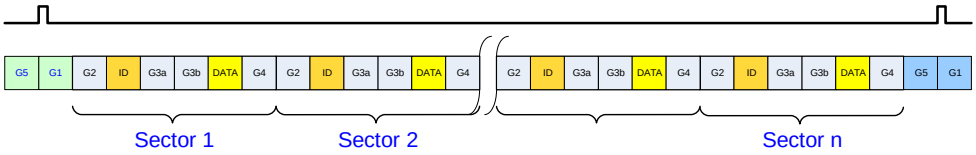
- **Description**: A “normal Atari diskette” has 80 tracks numbered 0 through 79 on each side. It is possible to write up to 82 or even 83 tracks on one side of a diskette. It is also possible to “hide” one or several tracks on the second side of an “officially” (as specified in the boot sector) single sided diskette.
- **Creation**: Easy to create on Atari. Note that some early Atari drives cannot position the head past track 79 and beware that using tracks over 82 has been reported to *damage* some floppy drives.
- **Detection**: You have to probe the diskette using FDC commands to check if some extra tracks exist on one side (probing 82 tracks is usually sufficient). For Single Sided diskette, you also need to probe for hidden track on second side.
- **Duplication**: Easy by software.
- **Emulation**: Just need to store information for the extra tracks.
- **Example**: Passengers on the Wind (Infogrames) uses tracks 80 & 81.

2.1.1.2 Missing tracks (NOT-UFT)

- **Description**: A “normal Atari diskette” has 80 tracks numbered 0 through 79 on each side. It is possible that not all of these tracks are formatted. For detail description of unformatted track please refer to Unformatted Diskette / Track / Sector. Note that it is also possible to hide data in an “officially” unformatted track.
- **Creation**: On a non-preformatted diskette you only format the tracks that need to be formatted! On a preformatted diskette you need to mimic unformatted tracks by writing, for example, some random data to those tracks without sync.
- **Detection**: A ***seek*** command with the *verify option* should fail on unformatted track. A ***read address*** should not find any sector.
- **Duplication**: If only the presence of an invalid track is tested then it is easy to reproduce by software. Placing hidden data in what looks like an unformatted track is usually difficult to detect.
- **Emulation**: The preservation file needs to flag missing tracks (e.g. indicating 0 sector).
- **Examples**: Barbarian (Track 74 – 79 missing), Run the Gauntlet (Ocean Software), Kick Off 2

2.1.2 Shifted tracks (SFT)

Normally the first sector of a track starts sometimes after the index and the last sector of the track end-up before the next index pulse. The pre-index GAP (at beginning of a track) is about 60 bytes and the post-index preamble GAP (at the end of a track) is about 600 bytes. In this case the track **write splice** (location where the floppy drive write gate is turned on/off) is located at the index.



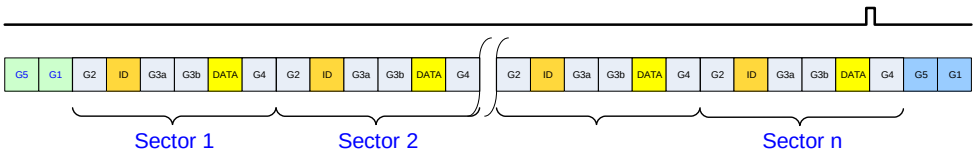
Sector positions relative to the index pulse for a normal track

Several protections shift the position of the track relative to the index. In this case the track write splice is not anymore located at the index. The shifted track protections can be further sub-classified as explained thereafter but usually this is not important for emulation.

*This type of protection is challenging for hardware copier. The copy should **not** be done from index to index as this will results in a track write splice in middle of data segment. The copy should start from the first sector until the last sector using the correct shifted starting position with respect to the index.*

2.1.2.1 Data over index (SHT-DOI)

- **Description**: A sector where the *Data Field* span "over the index". Normally all sectors of a track should end up before the index pulse. Yet it is possible to create a track with a total length that is slightly more than what a normal track can hold. This results in the last sector "wrapping around" the beginning of the track. As there is a small area at the beginning of a track (the post-index GAP) which is not used for storing data, it is possible to overwrite partially this section of the track.

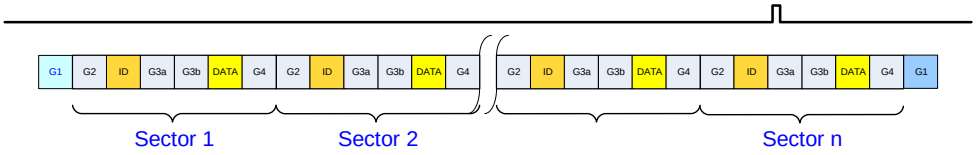


Sector positions relative to the index pulse for a track with Data over Index

- **Creation**:
  - On Atari: it is possible to create a "long track" with a total length that is slightly more than what a normal track can hold (usually about 10 to 20 bytes). This is done by placing the header of the last sector close to the end of the track. The ***write-track*** command starts at the index pulse and continues until the next index pulse. Therefore the last sector will be **truncated** during the format operation. However the ***write-sector*** command on this truncated sector will execute normally and this will result in data being written over and beyond the index pulse.
  - On Mastering machine: Normally writing a track is triggered by the index pulse. It is possible to shift the start of the write operation by a small amount (for example time of 20 bytes) and of course to shift by the same amount the stop of the write operation.
- **Detection**: The last sector spread over the index pulse but it is read as a normal sector by a ***read-sector*** command. It is therefore necessary to use a ***read-track*** command to find out that the last sector actually wrap over the beginning of the track.
- **Duplication**: Once detected the duplication of such sector can be done by formatting correctly the track.
- **Emulation**: Requires to store the track information in the preservation file.
- **Example**: Kick Off 2 places almost all the data of one sector at the beginning of a track.

2.1.2.2 Data beyond index-pulse (SHT-DBI)

- **Description**: This is an extreme variation of the Data over index protection. Normally all sectors of a track should end up before the index pulse but it is possible to create a track where the *ID Field* for the last sector is placed at the very end of the track with the corresponding *Data Field* placed at the very beginning of the track. You have to remember that the Data Address Mark of the *Data Field* is to be found within 43 bytes from the last *ID Field* CRC byte and therefore placement of the *ID Field* and corresponding *Data Field* in the track is needs to be very accurate. The last sector "wraps around" the beginning of the track. See Computer Hits Volume 2 for an example.



Sector positions relative to the index pulse for a track with data field beyond index

- **Creation**: It is almost impossible to position correctly such ID field on an Atari. Therefore this protection was usually created with mastering machines. The track is shifted so that the index pulse occur just at the end of the last ID field and of course the corresponding data field is located at the beginning of the track.
- **Detection**: This type of sector is read normally by the ***read-sector*** command. It is therefore necessary to use a ***read-track*** command to find out that the last sector actually spread over the beginning of the track.

> Note: The DMA can only transmit multiple of 16 bytes from the FDC. Therefore during a ***read-track*** command, one or several of the last bytes (always less than 16) may **not** be transferred by the DMA. Consequently it is possible that a ***read-track*** does **not** transmit the *ID Field* (or transmits it partially) when it is placed at the very end of a track. However the FDC ***read-address*** and ***read-sector*** commands will find this ID field for this sector correctly.

- **Duplication**: It is almost impossible to ***reliably*** place an ID field at the very end of the track on an Atari due to floppy drives rotation speed variation. Therefore this protection requires specific hardware to reproduce the key disk.
- **Emulation**: Requires to store the track information.
- **Example**: Computer Hits Volume 2 (Beau-Jolly)

2.1.2.3 ID over index (SHT-IOI)

- **Description**: A sector where the *ID Field* span "over the index". This is a variation of the Data Filed Over the Index-pulse protection. But in that case the index pulse happen inside an ID field. Please refer to the Data Field Over Index-pulse protection for details.
- **Creation**: It is almost impossible to position an ID over the index on an Atari. Therefore this protection could only be created on mastering machines.
- **Detection**: It is usually not possible to read this ID using a ***read track*** command because the ID segment is at the very end of the track and the data read usually get stuck in the DMA buffer (see above). Even though this ID can't be seen using a ***read track*** it can be read normally using ***read address*** and ***read sector*** commands.
- **Duplication**: It is almost impossible to ***reliably*** place an ID field at the very end of the track on an Atari due to floppy drives rotation speed variation. Therefore this protection requires specific hardware to reproduce the key disk.
- **Emulation**: Requires to store the track information.
- **Example**: Colorado, Computer Hits Volume 2 disk 2.

2.1.2.4 ID beyond index (SHT-IBI)

- **Description**: This is an extreme variation of the ID over index protection. In this case the sync marks that belong to the last ID field are located before the index pulse but the rest of the ID fields and the corresponding data fields wrap around the beginning of the track.
- **Creation**: It is impossible to position an ID over the index on an Atari. Therefore this protection could only be created on mastering machines.
- **Detection**: It is usually not possible to read this ID correctly using a ***read track*** command because the sync of the ID segment are located at the end of the track and therefore not seen by the ***read track*** command. Even though this ID can't be seen using a ***read track*** it can be read normally using ***read address*** and ***read sector*** commands.
- **Duplication**: It is impossible to place an ID field at the very end of the track on an Atari due to floppy drives rotation speed variation. Therefore this protection requires specific hardware to reproduce the key disk.
- **Emulation**: Requires to store the track information.
- **Example**: TODO

2.1.3 Track Layout Pattern (TLP)

- **Description**: With the WD1772 FDC it is possible to slightly modify the layout of a track by varying the number of characters in the gaps in different position of the track (e.g. vary the length of the GAP4 placed between the different sectors). It is therefore possible to create a track with a specific layout pattern different from the standard pattern. This is a sort of floppy disk ***water-marking*** technique.
- **Creation**: It is quite easy to format a track with specific values for each GAPs by sending the appropriate information to the FDC during the ***write-track*** command.
- **Detection**: Measure the layout of the different fields of the track using the ***read-track*** command and look for a specific pattern. Note that some tolerance needs to be taken in account as the number of bytes reported for a specific gap may vary from read to read.
- **Duplication**: Once detected it is easy to duplicate by software.
- **Emulation**: Requires storing the track information in the preservation file.
- **Example**: Not used on Atari?

2.1.4 Number of Sectors (NOS)

- **Description**: The standard Atari FD format uses tracks with 9 sectors of 512 data bytes. However many games use 10 or even 11 sectors per tracks just to pack more data on the diskette. The following number of sectors are often used:
  - ✶ Tracks with less than 9 sectors are not standard: They often combined sectors with 1024 data bytes. However alone they should not be considered as a protection.
  - ✶ Tracks with 11 sectors: it pushes several of the parameters that can be handled by the WD1772 FDC close to their limits. This is especially true considering that the Floppy Drive standard allows a 3% rotation's speed variation. These tracks are therefore often referred as **"read only"** tracks because once written they can't be modified. This is due to insufficient space/time available between the ID sector and the DATA sector (intra-gap space) for the write sector command to work correctly.
  - ✶ Tracks with 12 or more sectors: clearly indicate that some "tricks" have been used as 12 real sectors **won't fit** on a track.
- **Creation**: Easy in software. But remember that with 11 sectors it is almost impossible to write data **consistently** without using special hardware.
- **Detection**: Easy with multiple ***read-address*** command.
- **Duplication**: Easy in software for a number of sectors per track up to 10.
- **Emulation**: Requires nothing special the preservation file just needs to store the data information for all the sectors of the track using ***read-sector*** commands.
- **Examples**: Computer Hits Volume 2: 11 sectors / track, Theme Park Mystery: 12 sectors / track, Sherman M4: 70 sectors / track.

2.1.5 Sector Sizes (SSZ)

- **Description**: Normally the tracks have sectors with 512 bytes long *Data Field*. But it is possible to create a track with different data field size (usually a mixture of 512 and 1024)[1]. This is a more reliable approach to increase the overall capacity of a track rather than using 11 sectors of 512 bytes. Non-standard sector size are not be considered as a protection. Two common examples of format used are:
  - 9 sectors of 512 bytes plus 1 sector with 1024 bytes, and
  - 5 sectors of 1024 bytes plus 1 sector with 512 bytes.
- **Creation**: Easy on Atari.
- **Detection**: Easy with multiple ***read-address*** command.
- **Duplication**: Easy on Atari.
- **Emulation**: Requires nothing special the preservation file just needs to store the data information for all the sectors of the track using ***read-sector*** commands.
- **Examples**: Kick Off 2, Turrican uses tracks with a mixture of 1024 and 512 bytes sectors.

2.1.6 Invalid ID Field (IIF)

An *ID Field* contains the following information after the ID Address Mark: a **Track Number**, a **Side/Head Number**, a **Sector Number**, a **Sector Length**, and two **CRC** bytes.
To understand this protection you need to know that during a ***read-sector*** command when an *ID Field* is located on the disk, the WD1772 compares the Track Number of the *ID Field* with its internal Track Register. If there is not a match, the next encountered *ID Field* is read and a comparison is made again. If there is a match, the Sector Number of the *ID Field* is compared with its internal Sector Register. If there is no Sector match, the next encountered *ID Field* is read off the disk and a comparison is made again. If the *ID Field* CRC is correct, the *Data Field* is located and an internal register is loaded with the Sector Length.

2.1.6.1 Nonstandard IDAM (IIF-NSI)

- **Description**: The normal IDAM (ID Address Mark) used by the WD1772 is the character $FE which is sent after a sync sequence of 3 $A1 sync marks. An undocumented feature of the WD1772 is to accept the character **$FF** as well as **$FD** as an IDAM[2].
- **Creation**: During a ***write-track*** command it is possible to use $FF instead of the normal $FE IDAM character.
- **Detection**: As the ***read-address*** command and the ***read-sector*** command execute normally it is easy to hide the fact that a non-standard IDAM has been used. Detection can either be done with a ***read-address*** command. You have to check that an $FF character has been used instead of $FE in the *ID field*.
- **Duplication**: Once detected this protection is easy to duplicate.
- **Emulation**: Requires to store the track information as well as the address information.
- **Example**: Z-out

[1] Note that several of the BIOS calls will **not** work for sectors with size different than 512.

[2] Note that, in MFM, for the marks characters between $F8 and $FF the least significant bit is always ignored by the WD1772 and therefore : $F8 = $F9, …, $FE = $FF

2.1.6.2 Invalid track number (IIF-ITN)

- **Description**: A track that has one or several sectors with *ID Field*s that contains a track number different from the actual track number. In order for the *type I commands* (e.g. ***seek***) to succeed, on such a track, the verify bit has to be reset. Otherwise the FDC check that at least one sector has the correct track number. The ***read-sector*** command using "standard" parameters will also fail.
- **Creation**: Use ***write-track*** command with incorrect track number in *ID Field*.
- **Detection**: The ***read-sector*** command compares the track number of the *ID Field* with the track register if this matches it then compares the sector number of the *ID Field* with the sector register. If any compare operation fails the FDC retry 5 times then terminate the command with a record not found (RNF) error. Reading this kind of sector is possible but requires playing with the FDC registers (i.e. loading the track register with invalid value).
- **Duplication**: Easy by software
- **Emulation**: The preservation file should store the exact ID block.
- **Example**: Star Glider 2

2.1.6.3 Invalid head number (IIF-IHN)

- **Description**: An *ID field* with an invalid Side/Head Number (i.e. not equal to 0 or 1). Normally this field is supposed to be equal to the side you are reading however it should be noted that the WD1772 does not use this information.
- **Creation**: It is possible to write invalid values for the Side Number of an *ID Field* by sending the appropriate data to the FDC during a ***write-track*** command.
- **Detection**: Use a ***read-address*** command and compare the side value.
- **Duplication**: Can easily be done by software
- **Emulation**: The exact content of the ID field need to be saved in the preservation file.
- **Example**: Star Glider 2.

2.1.6.4 Invalid sector number (IIF-ISN)

- **Description**: During the format command the character loaded into the data register of the WD1772 is written to the disk. However the characters $F5 and $F6 are used to write respectively the *Sync Characters* $A1 and $C2 with a missing clock transition and the character $F7 is used to generate two CRC bytes. This implies that it is not possible to create a sector with an ID ranging from 245 through 247 ($F5-$F7). In fact the WD1772 documentation indicates that the sector number should be kept in the range 1 to 240.
- **Creation**: It is **not** possible to create a sector with an ID in the range of 245-247 with the WD1772 FDC and therefore creating such *ID Field* requires a **mastering machines**.
- **Detection**: Can easily be done with a ***read-address*** command.
- **Duplication**: Requires special hardware.
- **Emulation**: The sector with an invalid ID number is read as a normal sector by a ***read-sector*** command and stored in the preservation file like any other standard sector.
- **Example**: Dungeon Master (FTL Inc.) use a sector number of **247** ($F7) on track 0

2.1.6.5 Invalid sector length (IIF-ISL)

- **Description**: An *ID field* with an invalid Sector Length (i.e. not in range 0-3). Normally this field is supposed to take the value 0, 1, 2, 3 corresponding to respectively 128, 256, 512, 1024 data bytes size. As the WD1772 only uses the last three bits of the sector length information it is possible to write sector length value larger than 3. For example 0x03 and 0xFF are equivalent.
- **Creation**: It is possible to write invalid values for the Sector Length of an *ID Field* by sending the appropriate data to the FDC during a ***write-track*** command.
- **Detection**: Use a ***read-address*** command to get all the fields.
- **Duplication**: Can easily be done by software.
- **Emulation**: The exact content of the ID field need to be saved in the preservation file.
- **Example**: Star Glider 2 Z-Out.

2.1.6.6 Invalid ID CRC (IIF-IIC)

- **Description**: A sector that has a CRC error in the *ID Field*. This results in a sector that cannot be read by the ***read-sector*** command.
- **Creation**: Easy with the ***write-track*** command. For example by sending 2 normal bytes (e.g. $00, $00) at the end of the field instead of one "Write CRC" character ($F7).
- **Detection**: It is possible to read this kind of sector ID field with a ***read-address*** command and to verify that it has a wrong CRC. But it is not possible to read the sector with a ***read-sector*** command.
- **Duplication**: Can easily be done by software
- **Emulation**: Requires to store the complete track and address information in the preservation file.
- Example: Does not seems to be used on Atari

2.1.7 Duplicate Sector Number (DSN)

- **Description**: A track where, two (or more) sectors use the same sector number. Using blindly a ***read-sector*** command, for this duplicated sectors, result in reading randomly one of the two sectors based on current head position. In order to read a specific one, it is necessary to issue a ***read-sector*** command delayed by a specific amount of time from the *index pulse*. Usually, to facilitate the process, these two sectors are placed well apart (e.g. at the beginning and the end of the track). Sometimes the second ID field is not followed by a data field (no sector block protections).
- **Creation**: Easy in software.
- **Detection**: Easy by using ***read-address*** and/or ***read-track*** commands.
- **Duplication**: Easy in software.
- **Emulation**: The information for all sectors including the duplicate sector needs to be saved. In is also necessary to store the position of the sector in the track.
- **Example**: Night Shift uses a duplicated sector numbered 66 (the duplicated sectors also use the no data block protections).

2.1.8 Sector within sector (SWS)

- **Description**: During creation it is possible to place a new sector that overlap with a previous one. Therefore when reading these sectors we have the impression that the second sector is located within the first one. The layout of a normal sector contains the fields: GAP2, **ID Field**, GAP3, **Data Field**, and GAP4. With this protection, the included sector has its own GAP2, **ID Field**, GAP3, and **Data Field** placed inside the **Data Field** of the including sector. This is possible because during a ***read-sector*** command the sync mark detector of the WD1772 is turned off and therefore the included field are treated as normal data (not sync sequence recognized). A detailed explanation of this protection can be found in the Theme Park Mystery example. An even more complex variant is to have a sector within another sector which is itself located within another sector (SWSWS). Even with such a complex layout it is possible to read correctly the "included sector"! For an example of SWS-WS-WS look at Computer Hits Volume 2. When you read the data block of a sector the FDC disables re-synchronization. It is also possible to have an included sector that is shifted by one bit-cell from the including sector, this allow to read data bits as well as clock bits of the overlapped data field as in Turrican to check presence of NFA.
- **Creation**: Only possible in specific cases on Atari and therefore usually only possible to using mastering machines.
- **Detection**: The ***read-address*** command allow to read the ID field of the including and included sectors. The ***read-sector*** command reads the including sector beyond the start of the included sector because during a ***read-sector*** command the sync mark detector of the WD1772 is turned off. The following sector is read normally as if no including sector was placed before. Usually look for this protection when a track has a number of sector equal or exceeding 12. To confirm this protection you can use a ***read-track*** command. Another alternative is to check the data inside the including sector's *Data Field* and look for GAP2 followed by an *ID Field* etc. However beware that this will not always work due to the way the FDC works. For example it is not possible to find the ID and DATA field of sector 16 inside sector 0 of track 2 of Computer Hits Volume 2.
- **Duplication**: Require special hardware. Often combined with other protections like NFA.
- **Emulation**: Once the protection is detected the preservation program should store the track layout and the information about the including and following sectors.
- **Example**: Theme Park Mystery, Computer Hits Volume 2, Turrican, Nitro Boost Challenge

2.1.9 Non Standard DAM (NSD)

- **Description**: The normal DAM (DATA Address Mark) used by the WD1772 is either the character $FB for *normal data* and $F8 for *deleted data* which is sent after a sync sequence of 3 $A1 sync marks. An undocumented feature of the WD1772 is to accept the character $FC or F9 as a DAM (see also Non Standard IDAM).
- **Creation**: During a write-track command it is possible to use $FC or $F9 instead of the normal $FB or $F8 DAM character.
- **Detection**: As the ***read sector*** command execute normally it is easy to hide the fact that a non-standard DAM has been used. Detection can be done through a ***read track*** command where you have to look for a $FC/F9 character instead of $FB/F8 in the header of the *DATA field*. Note that the DATA Field reads with no CRC error.
- **Duplication**: Once detected this protection is easy to duplicate.
- **Emulation**: Requires storing the complete track in the preservation file.
- Example: No example found

2.1.10 Sector with No Data (SND)

- **Description**: A sector with an *ID Field* but not followed by a *Data Field*.
- **Creation**: on Atari it is quite easy to format a sector of a track with an *ID field* not followed by a *Data Field* using a ***write-track*** command.
- **Detection**: This kind of sector is found using a ***read-address*** command, but is not found using a ***read-sector*** command. This is because during the ***read-sector*** command the FDC expects to find a DAM/DDAM within 43 bytes from last *ID Field* CRC byte, if not the sector data is searched again for 5 revolutions and the command is terminated with the Record Not Found (**RNF**) Status bit set.
- **Duplication**: Can easily be done by software.
- **Emulation**: Requires storing the track information in the preservation file.
- **Example**: Night Shift uses duplicate sectors 66 both of them having No Data fields

2.1.11 Invalid Data CRC (CRC)

- **Description**: A sector that has a CRC error in its *Data Field*.
- **Creation**: Easy during ***write-track*** command by using the same mechanism as described in Invalid ID CRC.
- **Detection**: Can easily be done using a ***read-sector*** command. The data sector is *read normally* but the CRC error status bit is set at the end of the command.
- **Duplication**: Can easily be done by software
- **Emulation**: The content of the sector should be stored as normal but the CRC error indicator must be added to the preservation file.
- **Example**: Populous

2.1.12 Data Track (DTT)

- **Description**: This kind of track does not contains any **standard** ID / Data / Gap fields. The track is usually composed of a special Header followed by a Single Data field. In order to be read correctly the Header needs to use 3 $A1 sync mark, but beyond that the rest of the track can be anything. The only way to read the data record is to use a ***Read Track*** command. During a ***Read Track*** command the sync detector of the WD1772 is active at all time and therefore any MFM sequence of bits that contains 0x000101001 will cause a resynchronization. To avoid resynchronization an escape character (often 0x07 or 0x0F) is inserted whenever the input data contains this sequence. When the track is read the escape characters are removed to get back the original data. Note that sometimes this protection is combined with the Fuzzy Data Track protection.
- **Creation**: As the Data record can contains “invalid code” (i.e. code like 0xF5-0xF7) it can’t be written using a ***Write Track*** command. It is therefore mandatory to use mastering machines to write this kind of track.
- **Detection**: A ***Read Track*** command is used. The software looks for at least three 0xA1 then decode the rest of the Header and then read the data record. A checksum can be used to verify that the data record is read correctly.
- **Duplication**: Not possible in software requires special hardware.
- **Emulation**: For emulation it is necessary to save the complete content of the track as read by the ***Read Track*** command.
- Example: Maupiti Island (escape character 0x07), Golden Axe, Hot Rod, International Soccer (escape character 0x0F)

2.1.13 Hidden Data into GAP (HDG)

- **Description**: It is possible to write data into any gap. However the data are usually placed in the post ID Gap (Gap of 22 bytes) or in the post DATA Gap (Gap of 40 bytes) as well as in the pre and post index GAP (respectively 664 and 60 bytes on standard diskettes). See "copy me I want to travel" from Claus Brod for a complete explanation and some interesting examples. For example some variation of the Copylock protections from 1988 store key value in the last two bytes (usually 00) of the pre index gap (just in front of the SYNC char) ~~TODO check in Aufit~~.
- **Creation**: Extra data can be written into Gap only during the ***write-track*** command. It is recommended to use **Sync Marks** in front of the data to be able to read them correctly (although reading pseudo random value may be part of the protection).
- **Detection**: You need to use a ***read-track*** command to be able to read the inter-sector information. But it hard to find this information if you do not know what and where to look for. Therefore some heuristic need to be used (e.g. presence of sync marks into GAP).
- **Duplication**: Although it is difficult to detect, it is easy to reproduce with the ***write-track*** command.
- **Emulation**: Requires storing the track information in the preservation file.
- Example: Jupiter Masterdrive

2.1.14 Invalid Data in Gap (IDG)

- **Description**: During the format command character loaded into the data register of the WD1772 is written to the disk. However the characters $F5 and $F6 are used to write the Sync Marks and the character $F7 is used to generate of two CRC bytes. This implies that it is not possible to have a character ranging from 245 through 247 ($F5-$F7) inside any of the GAPs[3]. Reading these characters into GAPs requires using a ***read-track*** command. In order for these invalid characters to be read correctly with a ***read-track*** command they are usually preceded by one or several ***sync*** character.
- **Creation**: It is **not** possible with the WD1772 to write a character within the range 245-247 into any GAP. Therefore writing invalid character into GAPs requires mastering machines.
- **Detection**: Can easily be done with a ***read-track*** command.
- **Duplication**: Require special hardware.
- **Emulation**: It is necessary to save the complete content of the track.
- **Example**: Operation Neptune & Bob Morane .0 uses 0xF7 as gap bytes, Dragon Flight

2.1.15 Sync Mark in Data (SMD)

- **Description**: This is not a protection per se but it can be used as an indicator: During a ***read-sector*** command the *Sync Mark Detector* of the WD1772 is disabled but during a ***read-track*** command the *Sync Mark Detector* is active at all time. For specific sequence of data bits during a ***read-track*** the detector detects a $C2 sync mark resulting in a shift of the following bits/bytes. This "feature" can be used to hide some information inside a *Data Field* (see "copy me I want to travel" from Claus Brod for examples).
- **Creation**: You have to write a specific sequence of bits, known to create a **false $C2** sync mark, within a *Data Field* during a ***write-track*** command. Note that these sequences rely on a poorly defined $C2 Sync Mark and are well known and described in many places.
- **Detection**: Read with a ***read-sector*** command, then read with a ***read-track*** command and compare the returned data.
- **Emulation**: Requires storing the track information in the preservation file.
- **Duplication**: Easy by software.
- **Example**: Turrican (shift to clock bits)

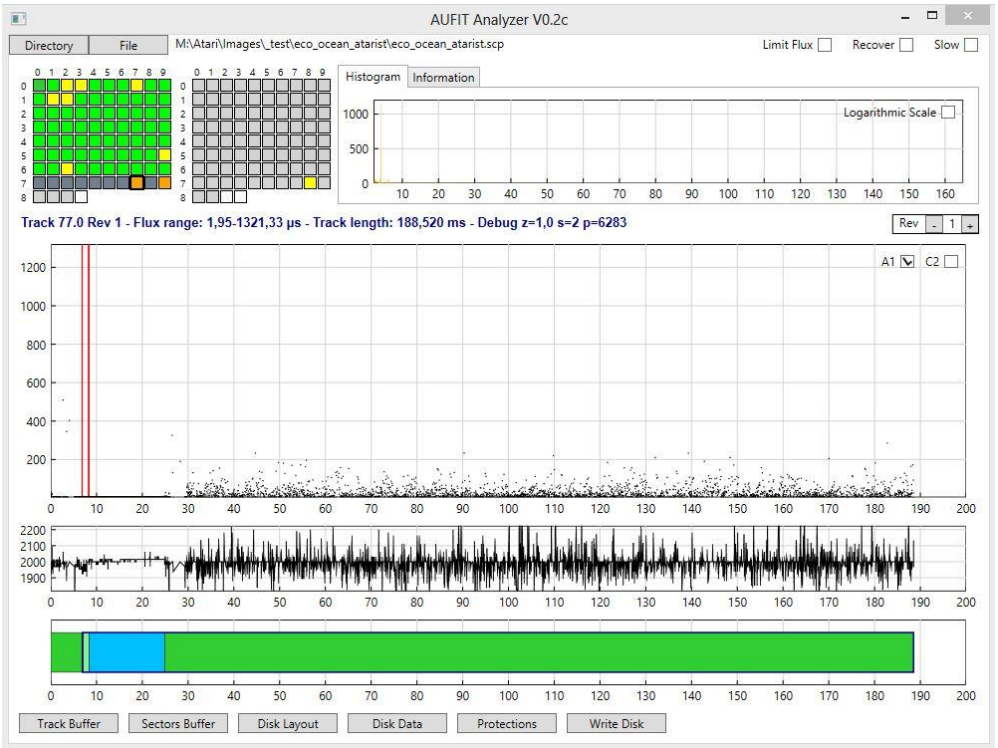
[3] Note that it is not possible to modify the GAP2 or GAP3b ($00). Therefore writing hidden bytes must be done in GAP1 and/or GAP3a and/or GAP4

2.1.16 Invalid Sync-mark Sequence (ISS)

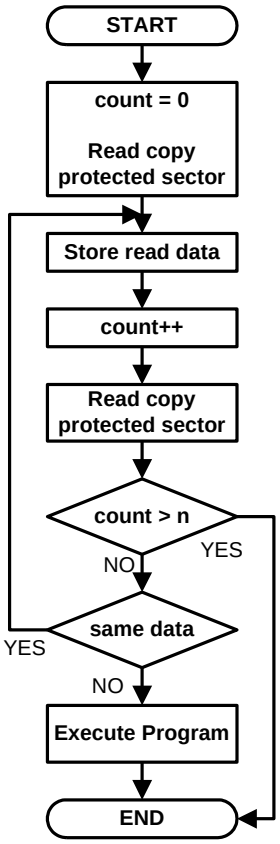
- **Description**: Normally Sync mark should always be in a sequence of 3 Sync Marks (3 $A1or 3 $C2) and should **always** been followed by an Address Mark (IAM = $FC, IDAM = $FE, DAM = $FB, or DDAM = $F8). Therefore having a sequence of 3 Sync Marks **not followed** by an AM is considered as an abnormal condition. Note that such sequence can usually be used to sync up the data separator to read data into gap or for the Data track protection. But it is also abnormal to have less than 2 or more than 3 Sync Marks in sequence. However finding only two Sync Marks with a ***read-track*** command is normal as the first Sync Mark is not read correctly.
- **Creation**: It is quite easy to create an invalid sync mark sequence during format by sending appropriate information to the FDC using the ***write-track*** command.
- **Detection**: Only possible with the ***read-track*** command as the ***read-sector*** command just ignore invalid sync mark sequences.
- **Emulation**: Requires storing the track information in the preservation file.
- **Duplication**: Easy by software.
- **Example**: Barbarian (one Sync alone on Track 0, series of Sync on Track 48 & 62)

2.1.17 Partially formatted track (PUT)

- **Description**: Inside what looks like an unformatted track it is possible to hide a sector.
- **Creation**: This kind of track can only be created using special hardware.
- **Detection**: The program verify that it can only reads the known sector and that no other sector exist.
- **Emulation**: Requires to store the content of the ***read track*** command in the preservation file.
- **Duplication**: Requires special hardware.
- **Example**: Eco



2.1.18 Fuzzy Sector (FZS)



- **Description**: A sector that contains **fuzzy bits**. Reading this sector several times returns different data.
- **Creation**: Cannot be created on Atari, requires mastering machines. Please refer to the **fuzzy bits** section**.**
- **Detection**: The flowchart on the right describes a copy recognition routine that tests for fuzzy bytes in the data field (patent 4,849,836). The protected sector that contains fuzzy bytes is read several times and randomness of the returned data is checked. If the same data is read several times on the protected sector the program is not executed. Very often, as in *Dungeon Master*, the protection is verified several times during execution of the game/program.
- **Duplication**: Difficult and requires special hardware.
- **Emulation**: The preservation file should have an indicator to record the fact that a sector has Fuzzy bytes. Usually the first and last 32 bytes of a fuzzy sector do not contain fuzzy bytes. It is also good to store information about bits that have changed in the different read operations.
- **Example**: TODO

2.1.19 Fuzzy Track (FZT)

- **Description**: This is somewhat similar to Fuzzy Sector: the protected track that contains fuzzy bytes is read several times and randomness of the returned data is checked. This is usually done in specific areas as explained below.
- **Creation**: Cannot be created on Atari, requires mastering machines. Please refer to the **fuzzy bits** section**.**
- **Detection**: If you know where to look for it is easy to read the same fuzzy data several times and to check that returned data are random. However detecting randomness in a read track without specific information is difficult because there are many reason why a read track returns random data. For example the head of the track until the first sync is random because the position where the read track starts vary, and the sector write splices also generates randomness.
- **Duplication**: Difficult and requires special hardware.
- **Emulation**: The preservation file should have an indicator to record the fact that a track has a Fuzzy data track. Note that Pasti STX does not support this kind of protection.
- **Examples**: Power Drift (track 1 side A of floppy disk 2). Vroom.

2.2 Protections based on timing

This section describes the protections based on variations of the standard 4 µs cell bit-rate. Although different techniques are used the end result of using bit-rate variation is always the same (with the exception of NFA): the overall time-length of a byte, transferred to/from the drive, is different from a "normal 32 µs byte". Therefore detection of this protection requires to be able to measure timing information when reading a block of bytes and/or a sector.

2.2.1 Long / Short Sector (LGS & SHS)

- **Description**: This kind of sector can be created by writing a sector of a track with an apparent rotation speed of the drive slightly above or below the normal speed. In practice this is obviously not done by varying the rotation speed of the drive but by changing the bit-cell clock. This results in a reading time for this sector above or below the reading time of a "normal sector". The IBM standard specifies that the FDC circuitry should handle a variation of the drive's rotation speed within ± 2% range. Therefore the DPLL of a FDC is supposed to accept at least a 4% variation. But in practice the WD1772 DPLL (See WD1772 DPLL Input Circuitry) can handle at least 10% variation for MFM encoding (as described in this DPLL Patent). It is therefore possible to write sectors with bit cells at frequencies between 225 and 275 KHz (corresponding respectively to 3.6 to 4.4 µs bit width) and to still be guaranteed to read the data correctly. However the resulting sector will be longer or shorter than a normal sector. The most famous usage of this protection was done by **R**ob **N**orthen in the **Copylock** (RNC) protection mechanism[4] (see an interview with Rob Northen): in this case the bit width is changed to approximately 4.2µs (about 4 to 5% variation) to result in a shorter sector. The beginning of the sector (for about 32 bytes) is written at normal speed so that we are sure that the data in this section are always read correctly.
- **Creation**: Cannot be done on an Atari. It requires mastering machines with the capability to vary the bit cell width on the fly.
- **Detection**: can't be done with standard TOS call. It requires to use specific routines to measure the time it takes to read the bytes in the short/long sector.
- **Duplication**: Difficult and requires special hardware.
- **Emulation**: The preservation file should store timing information about the sector.
- **Example**: Populous - Track 0 Sector 6. Back to the Future (T0S6 > 17200µs)

[4] According to vauvillf: there has been 2 RNC. The old one used for example on Arkanoid2, and Thundercats… It was possible to copy RNC-1 with the ***acopy*** program (only 2 to 3 times). Then there was a big evolution of the RNC protection sometime in 1988: with this one it was no more possible to copy the protection by software, and it was also using the famous trace decoding loop. Apparently the description provided here refers to the RNC-2 protection.

2.2.2 Long/Short Track (LGT & SHT)

- **Description**: This kind of track can be created by writing all sectors of a track with an apparent rotation speed of the drive slightly above or below the normal speed. This results in a track that contains more or less bytes than a normal 6240 bytes track. In practice this is obviously not done by varying the rotation speed of the drive but by changing the bit-cell width. The IBM standard specifies that the FDC circuitry should handle a variation of the drive's rotation speed within ± 2% range. Therefore the DPLL of a FDC is supposed to accept at least a 4% variation. But in practice the WD1772 DPLL (See WD1772 DPLL Input Circuitry) can handle a 10% variation for MFM encoding (as described in the DPLL Patent). It is therefore possible to write sectors with bit cells at frequencies between 225 and 275 KHz (corresponding respectively to 3.6 to 4.4 µs bit width) and to still read the data correctly.
- **Creation**: It requires special mastering machines that can vary the bit cell width on the fly.
- **Detection**: You can use a ***read track*** command. The normal track length is around 6240 bytes and it is sufficient to checks that the track has more (or less) than a 5% above the nominal value (e.g. less 6027 in Arkanoid).
- **Duplication**: Difficult and requires special hardware.
- **Emulation**: The preservation file should store timing information about the track as well as the number of bytes of the track.
- **Example**: Arkanoid , Indiana jones last crusade, Guntlet II, Garfield, speedball Awesome (T79 < 6000 bytes)

2.2.3 Intra-Sector Bit-rate Variation (IBV)

- **Description**: This is a more difficult to detect bit-rate variation. A sector is divided into several segments. Each of them uses a "drive rotation speed" slightly above or below the normal speed. By using faster and slower segments in the same sector it is possible to have the timing of these segments to compensate resulting in a sector with an overall normal timing. The only known protection of this type used in Atari is the **Macrodos** protection from *Speedlock Associates*. One sector is divided into 4 segments with normal-faster-slower-normal rotation speed resulting in a standard time length.
- **Creation**: Requires mastering machines that have capability to vary the bit width.
- **Detection**: It is quite difficult to detect this protection because the overall sector length is usually kept to a "normal" length. It is therefore necessary to measure the timing of block of characters (usually multiple of 16 using DMA transfer) inside a sector and to compare it to standard block length to check for specific above or below patterns.
- **Duplication**: Require specific hardware
- **Emulation**: The preservation file should store detail timing information about the sector. On Atari it is only possible to store timing information about reading a 16 bytes block.
- **Example**: Golden Axe, Colorado, Starblade, Treasure Trap, Damocles

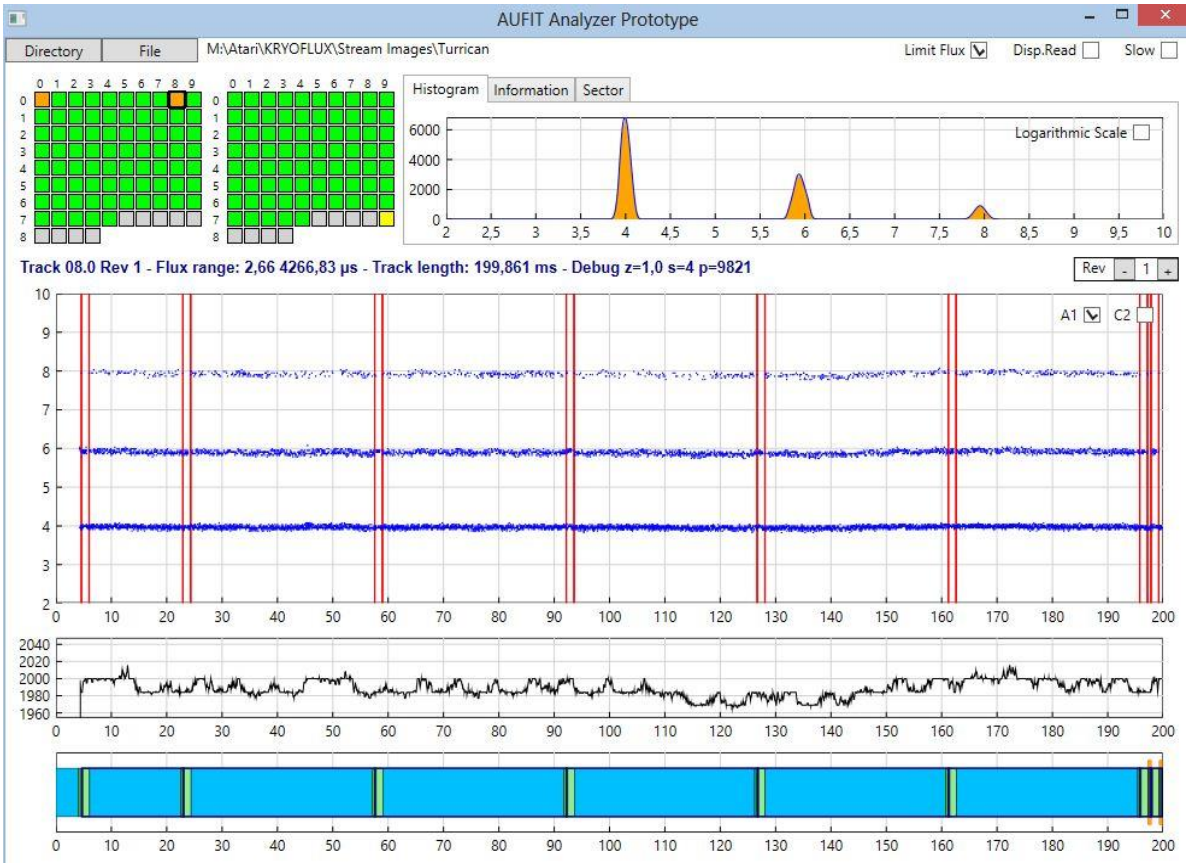
2.2.4 No Flux Area (NFA)

- **Description**: A track that contains a ***very long area without flux transition***. Note that this is quite different from an unformatted area (no flux transition recorded) because reading an unformatted area return random flux transition due to the fact that the gain of the amplifier (ACG) on the read channel is pushed to its maximum picking up noise on the head. In order to produce such area some trick needs to be used as explained in the No Flux Area on Disk section. This is extremely difficult to produce even with specialized HW.
- **Creation**: Requires specific mastering machines.
- **Detection**: No Flux Area result in reading 0x0000 MFM word in the shift register (no clock transition and no data transition). However only the data byte of the MFM word can be read by a WD1772 FDC and it is therefore not possible to directly check that the clock byte is also null. The only way to check the NFA protection is by using sector within sector where the included sector is shifted by a half-cell. The including and included sectors should contain the NFA area. The first sector allow to read the "data part" of the NFA and

the following sector allow to read the “clock part” of the NFA. For more information refer to Checking NFA with the WD1772 section.

- **Duplication**: Difficult and requires special hardware.
- **Emulation**: The preservation file need to save the track data and also needs to save the two sectors that allow to read the data and the clock.
- **Example**: Turrican.

Here is an example of a NO Flux Area that is located over the index. As indicated the NFA is 4.27ms long, starts before the end of the track, and wrap around the index.



Chapter 3. Preservation of Atari floppy disks

Information about **protection mechanisms** presented in this document should help in the creation of techniques/programs for **duplication**, **preservation**, and **emulation** of original Atari diskettes with the following philosophy:

> ✐ A preservation technique should always do the most to ensure the integrity of the preserved data. The preserved data should operate just like the original and not remove any protection, or modify the program being preserved in any way. The preservation technique must do the up most to check that the preserved data is identical to the original.

Specially designed programs can duplicate key disks for many of the protections presented here. But duplication of key disks using more advanced protections requires using specially designed hardware like the vintage ***Discovery Cartridge*** or the recently released ***KryoFlux*** and ***SuperCard Pro*** devices***.*** Analog hardware copiers, like the ***Blitz*** cable and associated software, can sometime create a working copy of a protected diskette but they **do not fulfill** the above requirements of producing a copy identical to the original.

Preservation has different meanings for different people but it can be classified into two categories:

- A “real preservation” is intended to save all the required information from a floppy disk so that it is not only possible to emulate the original FD but it is also possible to save all information used to create the disk and be able to physically duplicate the original FD. For example the files produced by the ***Discovery Cartridge,*** the ***KryoFlux,*** and the ***SuperCard Pro*** devices allow to emulate or to backup protected disks.
- An “emulation preservation” is intended to save enough information from a floppy disk so that it is possible to emulate the behavior of the original FD in a software or hardware emulator. For example the files produced by the **Pasti** imager allow to emulate protected disks.

It is interesting to note than most emulation / duplication programs do *not care* about (and sometimes can’t detect) the detailed underlying protection mechanisms used. They just store enough information to replicate the effect of a specific protection. For example they will detect fuzzy bytes but they will not care if they are the result of bits in Ambiguous areas, or bits rate violation. As a matter of fact finding the exact underlying causes often requires specific hardware.

In the following sections we are going to explain how to correctly use several devices specially designed to preserve Atari floppy disks.

3.1 Cleaning Rules to correctly image a floppy disk

Here are some basic rules to follow to create the best possible image:

- Use a known good original: Always use original disk and try to find out if the disk has not been modified.
- Write protect your original: In order to keep an unmodified disk always make sure that the original have the protect notch in the correct position at all time you use the disk including during the imaging operation.
- Clean your original: Atari floppy disks games are getting very old. They are prone to be dirty even if not used too much because of the environment. This results in deteriorated magnetic signal picked up by the read head. Carefully clean your disks with rubbing alcohol and cotton swabs. Rotate the disk in its jacket, cleaning the surface until no more residue is found on a clean cotton swab.
- Clean your floppy drive head: After reading several disks the head will have accumulated a lot debris. Clean the drive’s head with a commercial head cleaner or by using the same rubbing alcohol and cotton swab technique used to clean your disks.

3.2 Why do we need several revolutions for preservation?

You might be tempted to sample flux transition for only one revolution in order to save space on disk. However this is **not feasible** for preservation. Here is the rule:

- For duplication you can usually sample the flux transitions of only one revolution. If your original is in **perfect condition**, if you set all parameters of you mastering device correctly, and if the floppy you use for the backup is also in perfect condition then you should get a perfect backup of the original floppy.
- For preservation you **must** sample the flux transitions of **at least three revolutions**. But in order to be able to verify the integrity of the sampled data as explained below it is recommended to sample **five revolutions**.

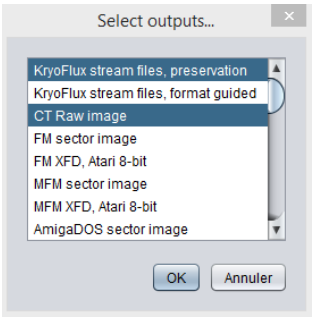
The rational for using five revolution is the following:

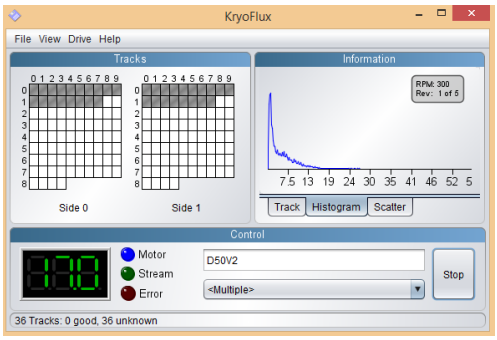
- By definitions fuzzy bits are detected by reading several times the same regions and comparing if the data is different. Therefore this kind of protection implies to sample at least two revolutions but three is preferred (majority rule).
- Many Atari games uses protections based on shifted tracks. In such a case the region “under” the index belongs to an ID or a DATA field and therefore it is not possible to start reading or writing data at the position of the index but this must be done at the location of the track write splice. Therefore this kind of protection implies to sample at least two revolutions. The combination of the last two requirements result in the necessity to sample at least three revolutions.
- Because of the age of the floppy disks, the magnetic signal picked up by the read head is often distorted. A program like AUFIT uses advance DPLL algorithm that allows to recover many imperfection on the read flux transitions but unfortunately this is not always sufficient. By sampling extra revolutions it is possible to combine data from multiple revolutions to recover the original information. For example Aufit is able to select and use a correct sector (one with a good CRC) among multiple revolutions whenever a sector is read incorrectly only on some revolution(s). The more revolutions you have the more chances you have to recover!

Therefore based on the above if you want to reliably preserve information from a floppy disk it is recommended that you use 5 revolutions.

3.3 Kryoflux

Using Kryoflux for preservation is pretty simple. Start the DTC GUI (kryoflux-ui.jar) and in the ***select output*** field of the control section select ***multiple.*** This open a new window and here select ***Kryoflux stream files, preservation*** and ***CT Raw image***. This will save the stream RAW files in a separate directory as well as the CTA raw file.





By default Kryoflux device settings are set to preserve 5 revolutions and sample up to the maximum track number.

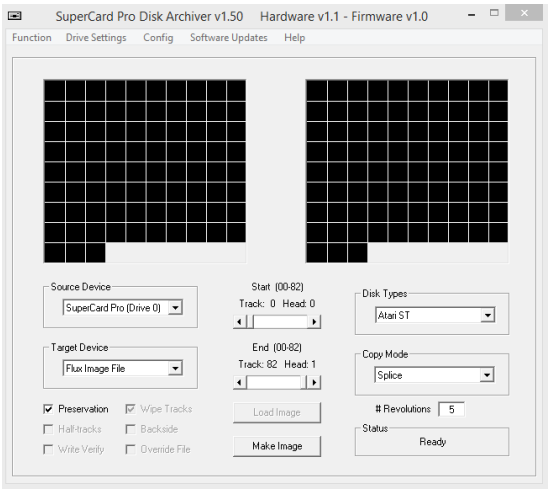
Therefore you just need to specify the image path (in file settings), the output name and start imaging.

3.4 Supercard Pro

Supercard Pro can be used to just backup (duplicate) Atari floppy disks or it can be used for real preservation. These two usages have different requirements:

- For duplication you can usually sample the flux transitions of only one revolution. If your original is in perfect condition, if you have selected the correct mode, and if the floppy you use for the backup is also in perfect condition then you should get a perfect backup.
- For preservation you must sample the flux transitions of **five revolutions** in splice mode.

For Supercard Pro you must change the # ***Revolutions*** value to 5 and the end track number to 82 **each time** you want to preserve a floppy as these values are not automatically saved (even using the ***save configuration*** command).

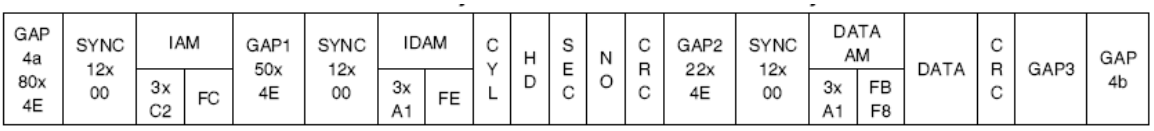


Chapter 4. Technical Information

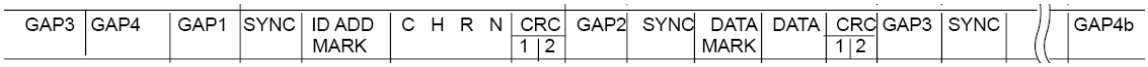
This chapter contains technical information that helps to understand in detail how data are written on floppy disks and how the some limitations of the WD1772 are used as protections.

4.1 Atari Low-Level Formats

The Atari ST uses the Western Digital WD1772 Floppy Disc Controller (FDC) to access the 3 1/2 inch (or to be more precise 90mm) floppy diskettes. Western Digital recommends to use the **IBM 3740 Format** for Single Density diskette and to use the **IBM System 34** Format for Double Density diskette. Actually the default format used by the Atari TOS is slightly different (closer to the ISO Double Density Format) as it does not have an **IAM** byte (and associated the associated GAP), before the first **IDAM** sector of the track (see diagram below). However the WD1772 (and therefore the Atari) is capable of reading both format without problem but the reverse is usually not true.

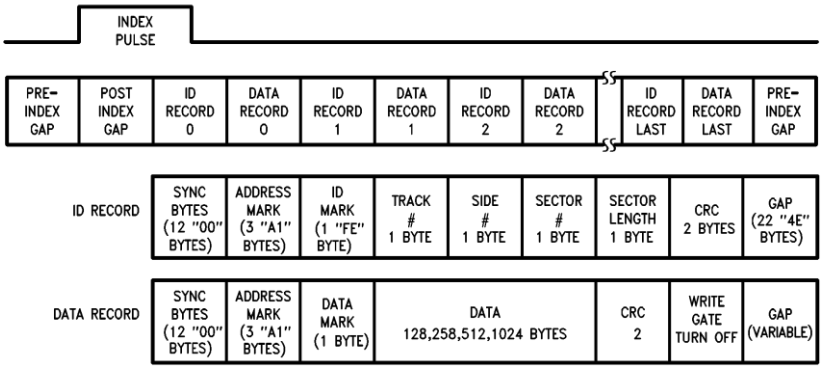


*IBM **System 34** Double Density Format (produced on a DOS machine formatting in 720K)*

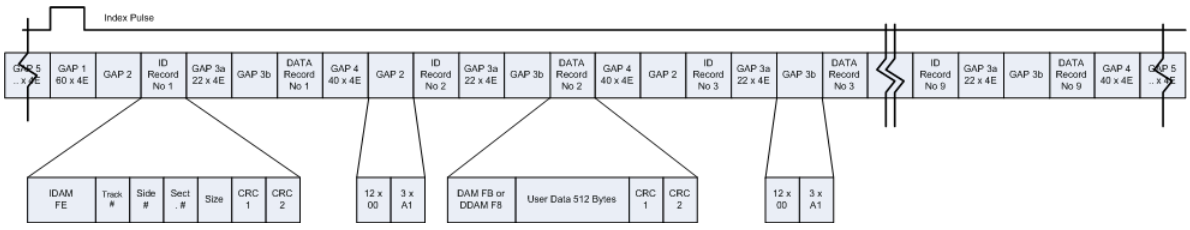


ISO Double Density Format.

Below is a detail description of the **Standard Atari Double Density Format** created by the early TOS.



Note: Many different conventions have been used to decompose and name the GAPS of a track. This document uses a GAP numbering scheme which is a combination of the IBM and ISO standards. It also decomposes the GAP between the ID record and the DATA record. Usually only one gap is described between these two records but in this document it is decomposed into an ID postamble gap (Gap 3a) and a DATA preamble gap (Gap 3b). This allows a more detail description, but of course they can be recombined into one more standard gap (Gap3). Although not shown in the diagram below a floppy formatted on an IBM has an extra IAM (index address mark) before the first sector. In that case the Gap1 is decomposed into two gaps: A post index gap (Gap1a) and a post IAM gap (Gap1b).



The table below indicates the standard values of the different gaps in the standard Atari diskette with 9 sectors of 512 user data bytes. It also indicates the minimum acceptable values (as specified in the WD1772 datasheet) of these gaps when formatting nonstandard diskettes.

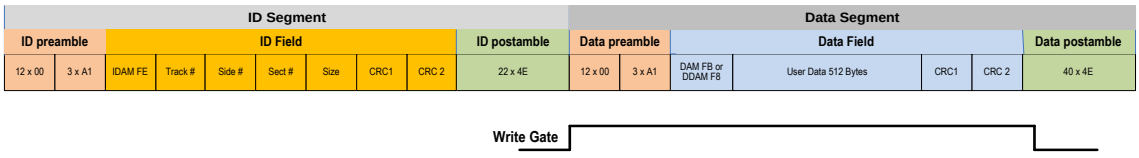
NAME	STANDARD VALUES (9 SECTORS)	MINIMUM VALUES (DATASHEET)
Gap 1 Index postamble	60 x $4E	32 x $4E
Gap 2 ID preamble	12 x $00 + 3 x $A1	8 x 00 + 3 x $A1
Gap 3a ID postamble	22 x $4E	22 x $4E
Gap 3b Data preamble	12 x $00 + 3 x $A1	12 x $00 + 3 x $A1
Gap 4 Data postamble	40 x $4E	24 x $4E
Gap 5 Index preamble	~ 664 x $4E	16 x $4E

Standard Sector Gaps Value (Gap 2 + Gap 3a + Gap 3b + Gap 4) = 92 Bytes / Sector
Minimum Sector Gaps Value (Gap 2 + Gap 3a + Gap 3b + Gap 4) = 72 Bytes / Sector
Standard Sector Length (Sector Gaps + ID + DATA) = 92 + 7 + 515 = 614 bytes

Note that the minimum values as specified in the WD1772 datasheet are not respected in the case of a track formatted with 11 sectors:
Minimum Sector Length (Sector Gaps + ID + DATA) = 72 + 7 + 515 = 594

The ID and DATA preamble are used to lock the PLL and should normally be kept as 12 $00 bytes. The FD format do not reserve a write splice byte (where the head write current is switched on or off) and therefore it should be considered as part of the data preamble field for format and write operations, and as part of the ID postamble for read operations.

One complete ID/DATA segment looks like this



As this format does not define any precise location *write splice* field, it should be included as part of the DATA preamble field for ***format*** and ***write*** operations and as part of the ID postamble for read operations.

4.1.1 "Standard" 9-10-11 Sectors of 512 Bytes Format

Note that the 3 1/2 FD are spinning at 300 RPM which implies a 200 ms total track time. As the MFM cells have a length of 4 µsec this gives a total of 50000 cells and therefore about 6250 bytes per track. The table below indicates possible values of the gaps for tracks with 9, 10, and 11 sectors.

Name	9 Sectors: # bytes	10 Sectors: # bytes	11 Sectors: # bytes
Gap 1 Index postamble	60	60	10
Gap 2 ID preamble	12+3	12+3	3+3
Gap 3a ID postamble	22	22	22
Gap 3b Data preamble	12+3	12+3	12+3
Gap 4 Data postamble	40	40	1
Total Gap 2-4	92	92	44
Record Length	614	614	566
Gap 5 Index preamble	664	50	20
Total Track	6250	6250	6250

Respecting all the minimum value on an 11 sectors / track gives a length of:
L = Min Gap 1 + (11 x Min Record Length) + Min Gap 5 = 32 + 6534 + 16 = 6582
(which is about 332 bytes above max track length). Therefore we need to decrease each sector by about 32 bytes in order to be able to write such a track. For example the last column of the table above shows values as used by Superformat v2.2 program for 11 sectors/track (values analyzed with a Discovery Cartridge).

As you can see the track is formatted with a Gap 2 reduced to 6 and Gap 4 reduced to 1! These values do not respect the minimum specified by the WD1772 datasheet but they make sense as it is mandatory to let enough time to the FDC between the ID block and the corresponding DATA block which implies that Gap 3a & 3b should not be shortened. The reduction of Gap 4 & 2 to only 7 bytes between a Data Field and the next ID Field does not let enough time to the FDC to read the next sector on the fly but this is acceptable as this sector can be read on the next rotation of the FD.

This has an obviously impact on performance that can be minimized by using sectors interleaving. But it is somewhat dangerous to have such a short gap between the data and the next ID because the writing of a Data Field need to be perfectly calibrated or it will collide with the next ID block. This is why such a track is usually reported as “read only” (as in DC documentation) and is sometimes used as a protection mechanism.
Of course you have more chance to successfully write 11 sectors on the first track (the outer one) than on the last track (the inner one) as the bit density gets higher in the latter case. It is also important to have a floppy drive that have a stable and minimum rotation speed deviation (i.e. RPM should not be more than 1% above).

4.1.2 “Standard” 128-256-512-1024 Bytes / Sector Format

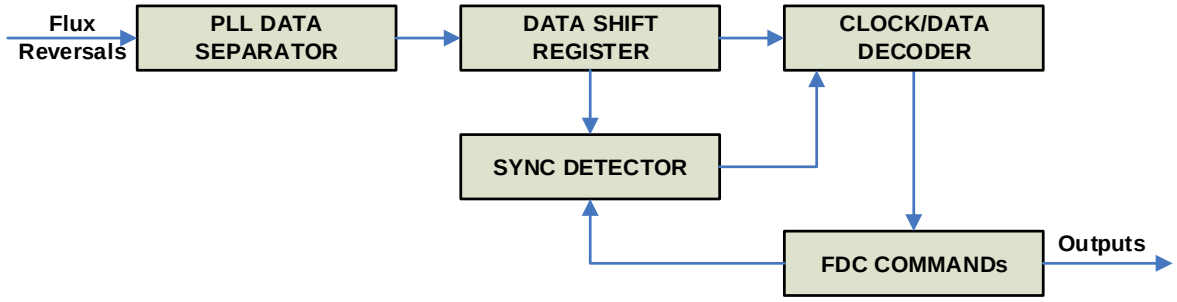
The table below indicates standard (i.e. classical) gaps values for tracks with sectors of size of 128, 256, 512, and 1024.

Name	29 sectors of 128 bytes	18 sectors of 256 bytes	9 Sectors of 512 bytes	5 Sectors of 1024 bytes
Gap 1 Index postamble	40	42	60	60
Gap 2 ID preamble	10+3	11+3	12+3	40+3
Gap 3a ID postamble	22	22	22	22
Gap 3b Data preamble	12+3	12+3	12+3	12+3
Gap 4 Data postamble	25	26	40	40
Total Gap 2-4	75	77	92	120
Record Length	213	343	614	1154
Gap 5 preamble	73	76	664	480
Total Track	6250	6250	6250	6250

4.2 WD1772 DPLL Input Circuitry

This section provides basic information on the DPLL of the WD1772 and how the decoded bits are entered into the FDC shift register. It does not describe the *data separator* which is based on usage of an AM (Address Marks) detector to find a specific pattern in the shift register (usually during gaps) as it is pretty simple to understand and not useful in the context of this document.

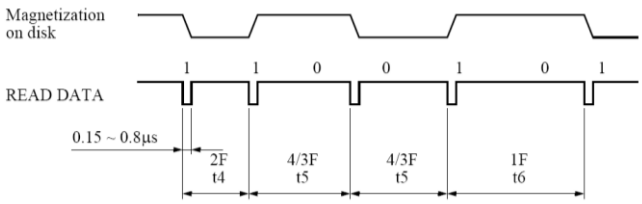
This is a simplified block diagram of the input circuitry of the FDC:



The WD1772 uses a digital phase lock loop (DPLL) circuit for reading the input data transmitted from FD media. Inspection windows are established that have duration proportional to the frequency of arrival of the data, and **start/stop times** that can be adjusted so that subsequent data bits will be received in the **middle** of the inspection windows. To achieve this, the DPLL circuitry applies ***frequency*** and ***phase*** corrections that compensate the input data frequency drift. This drifts are usually due to unsteadiness of the motor drive speed (the frequency drift), and the migrations of the magnetic reversals area (the phase drift). The DPLL used inside the WD1772, as well as many other FDC build in the 80s, implements an algorithm described in the public US patent 4,870,844. The patent is rather complex and in this section I will only highlight some of the most important aspects of the DPLL algorithm that are useful to understand the behavior in the context of fuzzy bits, long/short track, etc.

If you want to fully understand the behavior of the DPLL please refer to the patent. Note that in order to provide precise results my ***Aufit***, ***Analyze***, ***KFAnalyze***, and ***KFPanzer*** programs ***fully implement the DPLL algorithm as described in the patent***.

Typical MFM encoding

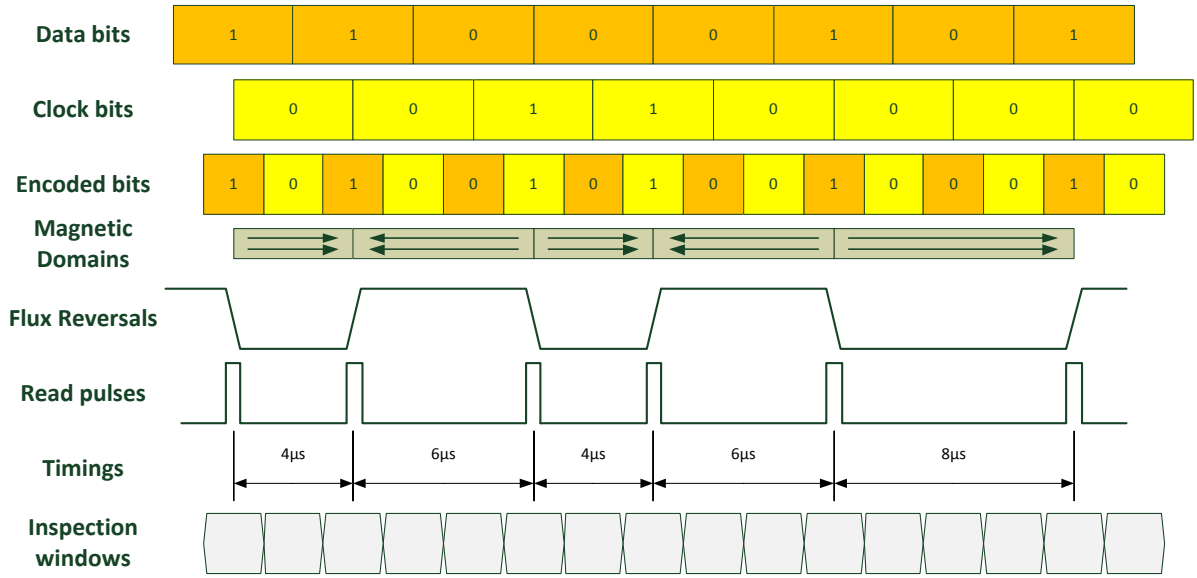


Note : READ DATA pulse will be detected within t7 from is nominal position. (When PLL separator is used with recommended write pre-compensation.)

Density mode	rpm	t4	t5	t6	t7
2MB mode	300	2µs, Nom.	3µs, Nom.	4µs, Nom.	±350ns
1MB mode	300	4µs, Nom.	6µs, Nom.	8µs, Nom.	±700ns

As we can see the **nominal values** for the possible reversals spacing in **DD MFM** (1MB mode) are: 4µs, 6µs, or 8µs.

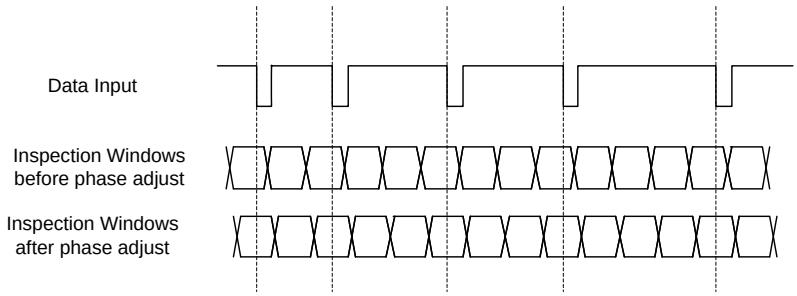
Let's first review a typical Double Density MFM data encoding:



The data input circuit of the FDC ensures that the data pulses received are converted into data bits and stored in the **data shift register (DSR)**. For that matter the digital phase lock loop defines *inspection windows* that repeat every 2µs (a half cell size). A one is input to the shift register if a data pulse is received at any time during one inspection windows; otherwise a zero is stored in the shift register as the value for the current bit.

The period of the inspection windows is gradually adjusted (expanded or shortened) to compensate an eventual frequency shift affecting the input data transfer. This frequency correction is computed based on the **history** of the location (relative to the inspection window) of the **last three** flux reversals.

Ideally, individual pulses should be located in the middle of the inspection windows. To achieve this, the start and stop times of the inspection windows are adjusted to compensate for deviation (from ideal) in time of arrival of the most recently detected data pulse. This phase correction is done proportionally to the distance of the reversals with the middle of the inspection window.



The proper ratio of phase and frequency correction provided in the loop is carefully balanced so that the DPLL is **fast settling** but **stable**. A large amount of phase correction cause the loop to settle faster but also make it more sensible to noise. On the other hand if too much frequency correction is used, the loop can become unstable.

It is interesting to note that the DPLL as defined in the patent allow an input frequency variation of up to **9%**. This corroborates the actual measurement made with a WD1772 that correctly interprets bits with a variation of at least 9 to 10 % for DD MFM (and about 100% for SD FM!). Note that these values are well above the variation used by the **Copylock** and **Macrodos** protection mechanisms (usually less than 5%) and therefore the data within this kind of sector should be read correctly.

4.3 WD1772 Detection of Border Bits

With the above information it is now easy to understand that if a bit reversals happens close to the border of an inspection window (also called Ambiguous area) it will be detected into the first or the next inspection window based on small variation of the drive rotation speed between two ***read-sector*** commands and this will therefore result in pseudo random values returned (fuzzy bits).

For example having a reversal 5µs apart from the previous one can be interpreted as a reversal after 4µs or a reversal after 6µs based on small frequency fluctuation of the rotation speed between two reads. One might argue that it is not possible to make sure that these "marginal reversals" will be positioned correctly due to the fact that the rotation's speeds of different drives are somewhat different and therefore precise reversals timing on a floppy diskette cannot be guaranteed. But in practice this is where the frequency and phase correction of the WD1772 DPLL comes into play. As explained above the inspection window will have it size (i.e. frequency) and position **corrected** based on the input reversals stream after reception of only a few reversals. Therefore the DPLL of the FDC automatically adjust the frequency of inspection windows for any acceptable (about 10%) variation of drive speed and adjust the phase so that a "normal reversal" will be perfectly in the middle of the inspection window and a "marginal reversal" will be perfectly at the border of the inspection window.

This also explains why, in most cases, "fuzzy bits" are used in "compensating pair": for every two subsequent fuzzy bits the first reversal is placed at one extreme (e.g. at the beginning) of the inspection window and the "compensating reversals" of the next fuzzy bit at the other extreme (e.g. at the end) of the inspection window. By using this kind of "compensating bits" we guarantee that the frequency and the phase of the inspection windows are (almost) not affected.

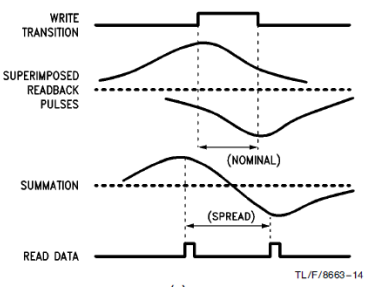
4.4 No Flux Area on Disk

There has been a lot of debate around the so called **No Flux Area** on disk. This is a protection's mechanism used on some floppies that results in absolutely **no flux transitions** coming from the drive read circuitry for a long period of time (usually several milliseconds).

For some times it has been thought that this was obtained by doing a so called "strong erasure" of areas of a disk. However this would be very difficult to create and it would not produce the wanted effect:

- For one this can't be done with the normal recording head/circuitry of a floppy **drive** and therefore it would require to use modified drives.
- Secondly if such areas, with no magnetic flux transitions, existed on the floppy disk it would cause the ACG of the read chain to be set to its maximum amplification value and this would result in picking up noise from the head resulting in reading random flux transitions which is not the case.

The following explanation of the **No Flux Area** has first been described by **István Fabián** from SPS (see the reference section) and can be summarize as follow:



Bit-shift occurs on ***any*** NRZ recorded medium as a normal consequence from read/write head operation. Data are written when the read/write head generates a flux change in the gap of the head, which causes a change in magnetization of the medium oxide. In reading, a current is induced into the read/write head when a flux transition on the medium is encountered. The current change is not instantaneous, since it takes a finite time to build up to peak and then to return to zero. If flux transitions are close together, the current buildup after one flux transition then declines, but it does not have time to reach zero before the second

transition begins. Consequently current pulses are summed by the read/write head, which causes the peaks to be shifted. A No Flux Area is created by writing a large number of flux transitions close enough (i.e. at a relatively high frequency). This will result in having the read current never returning to zero and consequently this will result as no data pulse generated on the read channel. Note that in this case the ACG is set to a normal amplification as the input circuitry receives high frequency flux transitions even if no data is coming out of the read channel.

4.4.1 Checking NFA with the WD1772

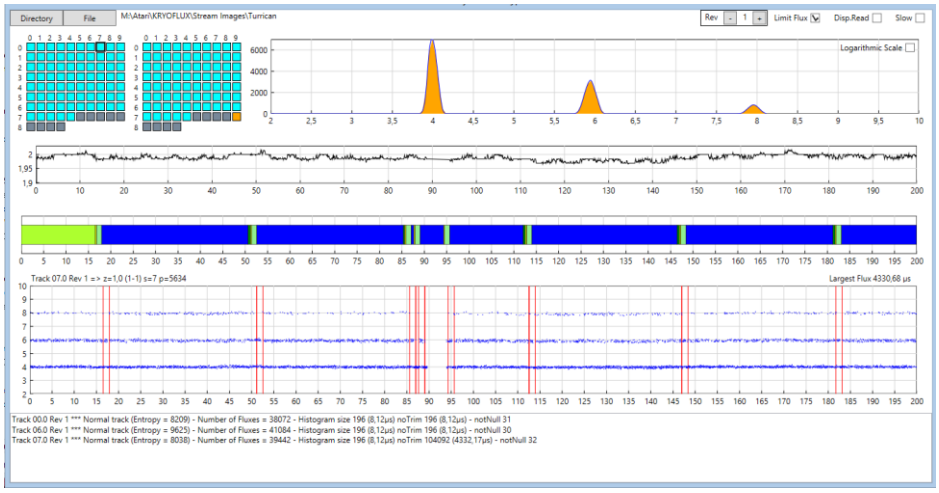
The next challenge is to check an NFA with a standard WD1772 Floppy disk controller? Normally the WD1772 FDC can only read the data bits. Therefore a sector with NFA is read as a sector filled of 0x00 bytes but it is normally not possible to check that the clock bits are also 0x00bytes. To be able to check the clock bits the NFA protection uses an interesting trick. Another sector is written within the first sector (SWS) that contains the NFA and this sector contains 3 sync marks shifted by one bit cell. Therefore when you read the data for this second sector you are actually reading the clock data from the first one!

Here is a dump made with **KFAnalyze** of the game Turrican where the data bytes are displayed on the first line of the dump, and the clock bytes are displayed on the second line.

```
Detail buffer content for sector 0 with 1027 bytes
= DATA ID=0 1027 bytes @87082 us length=32637.85 CRC BAD CLK=3.97 TMV=0 BRD=1 DOI=0
  *** Fuzzy Sector *** starting at byte position 217
  0000 87082  4000  fb 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
                    00 7f ff ff ff ff ff ff ff ff ff ff ff ff ff ff  .□..............
  0010 87596  3968  00 a1 a1 a1 fe 07 00 10 03 bb 21 4e 4e 4e 4e 4e  ..........!NNNNN
                    ff 0a 0a 0a 00 f8 7f e7 fc 00 4e 10 90 90 90 90  ......□...N.....
  0020 88103  3968  4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e  NNNNNNNNNNNNNNNN
                    90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
  0030 88614  3968  4e ff ff ff ff ff ff ff ff ff ff fe 14 14 14 14  N...............
                    90 00 00 00 00 00 00 00 00 00 00 00 00 a1 a1 a1  ................
  0040 89124  4000  00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
                    fb 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

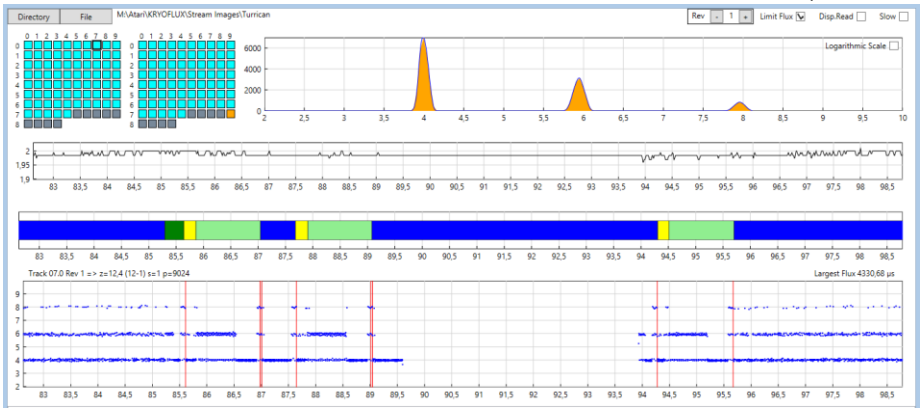
We can see inside data block (starting with 0xFB DAM) the presence of 3 sync character followed by the ID block for sector 16 (sector within sector). However if we look further down we do not see the sync marks for the corresponding data block. Instead we see the presence of 3 bytes with value $14 followed by a byte $00 and several bytes 0xFF. But if we look at the line below (that contains the clock bytes) we can see that the 3 x $A1 sync bytes are in fact located in the “clock” bytes. During the read command the sync mark detector of the WD1772 will take care of shifting the input stream by a half cell to correctly read the sector 16 data.

The end result is that you can read the “data” bytes of the NFA by reading sector 0, and you can read the “clock” bytes of the NFA by reading sector 16 (sector within sector).



As you can see around 90 ms inside the track we have a region without any transition for a period of 4330µs.

If we zoom close to the NFA we see that we have a first sector, and inside this sector we have a second sector (sws) and that the data segments of these sectors both includes the NFA.



Note: Inter-GAP in Green, ID in yellow, Intra-Gap in light green, Data in blue.

4.5 Unformatted Diskette / Track / Sector

4.5.1 Presentation

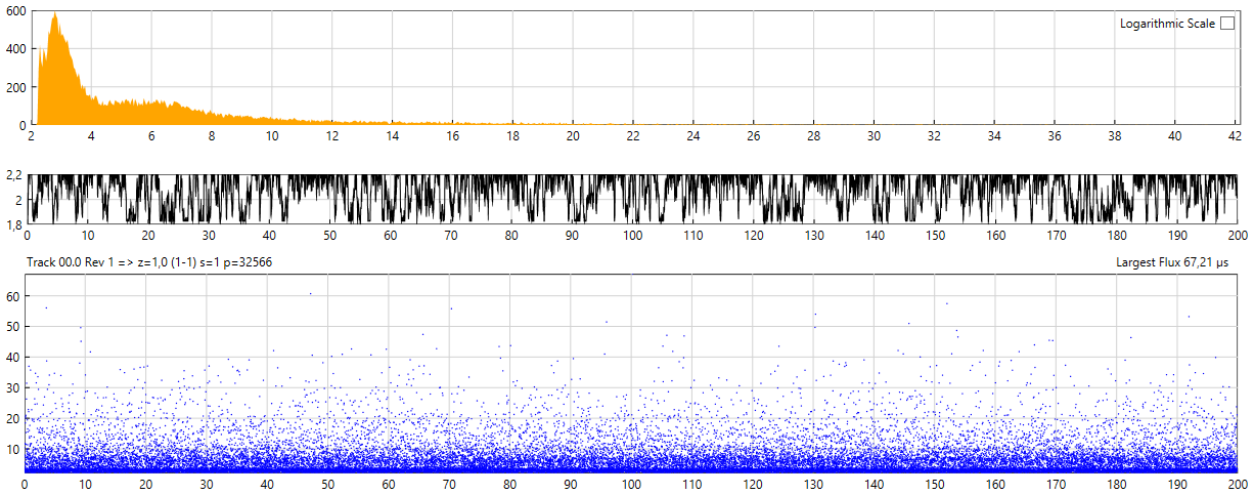
By definition an unformatted diskette would be a diskette that has never been formatted. During formatting, the particles are aligned forming a pattern of magnetized tracks, each broken up into sectors, enabling the controller to properly read and write data. Here is a definition from Wikipedia: *“A blank "unformatted" diskette has a coating of magnetic oxide with no magnetic order to the particles”.*

The magnetization on the surface should be relatively uniform and in an ideal world the head should not peak up any flux reversal and therefore the read circuitry should not return any data pulse. But in practice many pseudo random transitions are detected. Two things explain this behavior:

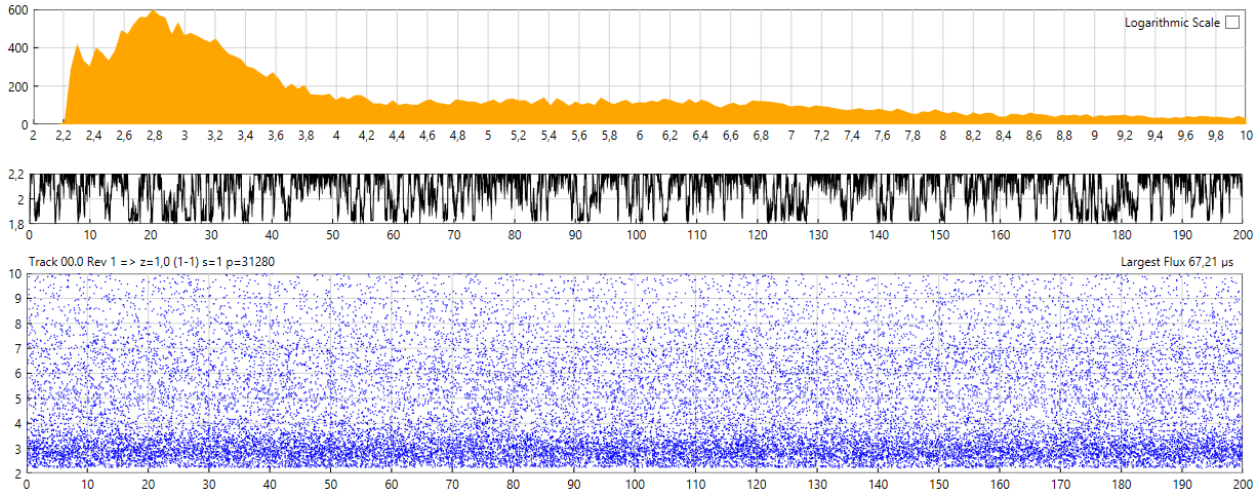
- As we have no regular flux transitions, the drive's automatic gain control (ACG) is pushed to its maximum possible gain because it presumes a weak signal coming from the drive's read head.
- Some random flux variations exist naturally on the magnetic surface and due to the fact that the ACG is turned to its maximum they may be detected as flux transitions. This can be compared to “hearing” noise from a blank audio tape when the volume pushed very high.

The detected data seems "random" in the sense that the data is never the same twice. But it seems that the “repartition” of these transitions is related to the drive speed (and humidity, temperature) fluctuations. Note that compared to a normal track the number of flux transitions detected on an unformatted track is obviously much lower.

We can see that most of the transitions are typically located between 2µs and 7µs. The image below show a typical histogram for an unformatted track:



If we limit the scale of the displayed transitions to 10µs we have the following image:

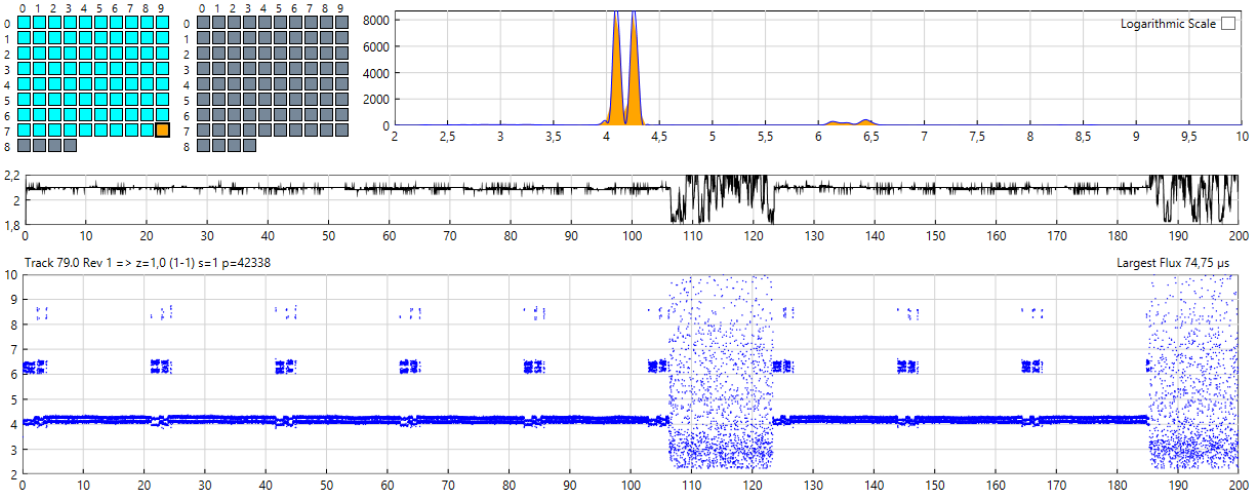


Few things I have noticed:

- Usually all the unformatted tracks of one diskette looks very similar (i.e. they have equivalent histograms) but they look different from one diskette to another. This “signature” is probably dependent of the diskette as well as of the drive used.
- When you record the flux transitions of a track over several revolutions, usually you do not see much difference between one revolution and the next except for unformatted track. Due to the random nature of unformatted track the data information recorded differ widely from one revolution to the next even though the histogram stay relatively the same.

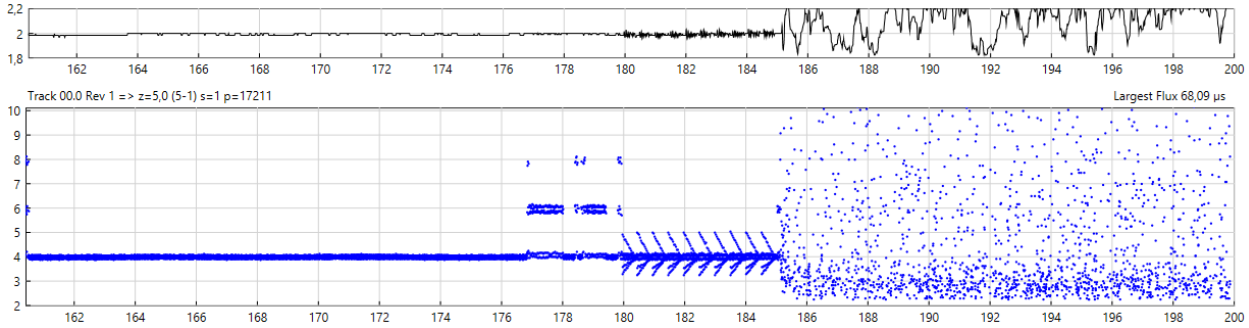
4.5.2 Partially unformatted track

Some protections use partially unformatted tracks. For example:



The example above shows the track 79.0 of Night Shift game. We can see that we have two unformatted sections in this track. The clock, decoded by the DPLL, in these regions vary a lot and you can easily imagine that the data read from these sections contains fuzzy bits.

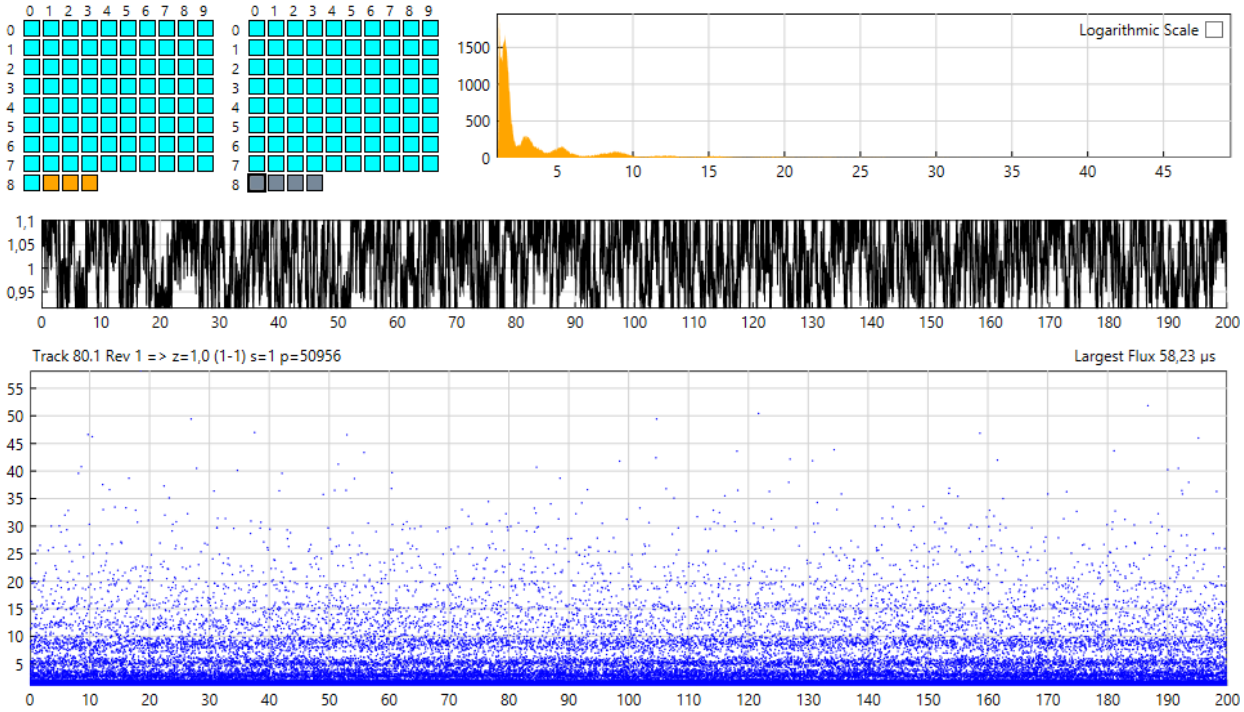
Another example taken from a protected diskette from DrT a D50 Patch Editor. In the following chart I have zoomed on an unformatted area near the end of track 00.0:



We see that the end of the track is unformatted and have therefore random transitions, but just before that area we also see a strange “fish bone” pattern. We have some flux transitions placed closed to 5µs and 3µs which are exactly at the border of the inspection window and this will certainly results in fuzzy bits. Note that with this fish bone pattern the transitions close to 5µs are “compensated” by transitions close to the 3µs and this results (thanks to the DPLL) in a relatively stable 2µs clock.

All the previous examples where taken from Double Density Floppy disks (the one used on Atari). But I have also experimented with High Density floppy disks (PC diskettes).

I have noticed that the unformatted tracks from a HD floppy look different! They seems to exhibit a much more pronounced “banding effect”:



This is probably due to usage of different magnetic coating that have different coercivity.

4.5.3 Partially formatted Track

Another trick used for protection is partially formatted track: At first glance the track seems to be unformatted but in fact it contains some information (sometimes this information is used by duplicators and/or by manufacturer). Some protections may use as little as just a few bytes of data and leave the rest of the track unformatted. Most copier won't be smart enough to detect this kind of hidden information.

However in order to use this trick for protection you not only need to write the hidden bytes but you also need to be able to read and check them correctly with the FDC of the platform. In the case of a MFM DD diskettes used with Atari ST, the only reliable way to decode and check hidden bytes in an unformatted track is to have one or several SYNC marks in front of the data. Even with sync mark it is difficult to detect this kind of information using a read track command unless you know exactly what you are looking for.

4.5.4 Unformatted track detection

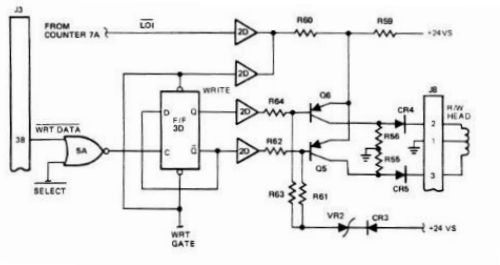
It is usually easy to visually detect that a track is unformatted on a scatter chart but it is more difficult to detect this by software. The detection should provide reliable information and ideally detect partially unformatted track as well as partially formatted track. There are several possible proposed indicators:

- If you cut a track in small chunk of data you should be able to match each of these blocks from one revolution to the next. Of course you need to allow for some slack in position from revolution to revolution.
- You can check for legitimate encoding on the track. For example in the case of MFM encoding this could be as simple as detecting SYNC marks sequence.
- I have successfully used a modified computation of the Shannon Entropy on the flux transition's histogram. A "normal track" has a coherent repartition and this result in a high entropy.
- The number of transition is lower than normal track.
- Unformatted track contains invalid MFM sequence. For example 0x0000

By combining the above indicators you should be able to detect "true" unformatted track (i.e. with zero encoded bytes).

4.5.5 How to reproduce unformatted areas on Floppy Disks?

The simplest idea that comes to mind is that we use a blank diskette and for unformatted areas we do not write anything. Unfortunately DD floppy diskettes you buy in the commerce are already preformatted (usually using the IBM format and with a DOS 2.0 boot sector). I don't know why the manufacturers do that, but I guess this must be for them to run some quality tests?



Note that **only** tracks 0 to 79 are preformatted on blank diskettes. If we sample the flux transitions of unformatted tracks beyond this limit (track 80-83) we can see that these tracks are similar but slightly different than unformatted tracks in range 0-79.

This seems to indicate that professional duplication machines might have used unformatted diskettes?

Writing unformatted track it is as simple as enabling the write gate and keeping the write data line of the drive to zero for the time of the track. This will keep the current flowing in the read/write head constant and therefore the flux will written will be constant.

The only potential problem comes from potential misalignment of the write head relative to previously written data. Hopefully the tunnel or straddle erase head should take care of the leftover of previous contents.

Remember that writing data on a floppy drive is different from the same operation on an audio tape drive. On an audio tape the information the data is first erase by an erase head then the new data is written linearly by the read/write head. On a floppy drive there is no erase head (other than the tunnel / straddle erase head used from trimming track) and the new data are just written over the existing one with the head operating at magnetic saturation.

To summarize: unformatting a track only requires to keep the write data line negated during the complete write operation. This is obviously not possible with a WD1772 FDC (it always pulse the write data line) but it is easy to control on a replicator (e.g. trace) or with a Kryoflux, SuperCard Pro device.

On an Atari the best you can do is to format a track using a buffer containing random bytes, but you will never get something similar to a real unformatted track because flux will always be located in the 4, 6, and 8 µs bands.

4.6 Fuzzy Bits

Fuzzy bits are known under many different names: *weak bits*, *wandering bits, flaky bits, flakey bits, phantom bits*, etc. ***Weak bits*** is the most commonly used term, however I find it confusing (as there is usually no "weakness" in weak bits). Therefore I prefer to use the term **Fuzzy bits** that does not indorse any underlying cause but clearly indicate the "fuzziness" nature of the returned data. Although fuzzy bits can be created by using different techniques the result is always the same: reading a byte that contains fuzzy bits will return random values (i.e. different values each time it is read). Fuzzy bytes could potentially be located at any place in a track but fuzzy bytes are often placed in the data field of a sector. To provide complete information we will describe below several ways to create fuzzy bytes: Flux reversals in Ambiguous Area, MFM Timing Violation, or Weak Bit. However for emulation or backup purpose it is not necessary to know underlying mechanism used.

4.6.1 Flux Reversals in Ambiguous Area

- **Description**: These fuzzy bits are obtained by "placing" certain flux reversals in so called "Ambiguous areas" i.e. *at the border of the inspection window*. Please refer to WD1772 Detection of Border Bits section for more information.
- **Creation**: These fuzzy bits are obtained by placing the bit flux reversals in "Ambiguous areas". More precisely the bit reversals are placed in locations that will confuse the DPLL (Digital Phase Lock Loop) of the data separator resulting in random values read (i.e. sometimes 0, sometimes 1). This is obtained by positioning the bit reversals at the **border of the inspection window**. In that case the data separator will return random values due to small variation of the drive rotation speed. In the US patent "Copy Protection for computer Disc 4,849,836" one of the techniques to create fuzzy bits consists in having flux reversals gradually sliding in and out of the inspection window border. Of course creating this kind of reversals requires special hardware that has capability to vary the FDC clock on the fly, or the capability to directly control the bit cell width/position (e.g. the Discovery Cartridge, KryoFlux board, SuperCard Pro device).
- **Detection**: As mentioned this protection results in Fuzzy Sector. Therefore it can be detected by reading the same fuzzy sector (i.e. sector that contains fuzzy bits) several times and checking that returned data are random. Without specific hardware it is not possible to find the real underlying cause of the fuzzy bits but this information is of no use for an emulator or a duplicator.
- **Duplication**: Difficult and requires special hardware (i.e. the Atari WD1772 cannot be used to copy this kind of bytes). Analog or digital copiers can be used but, as usual, digital copier should be preferred.
- **Emulation**: The preservation file should have an indicator to record the fact that the sector is a fuzzy sector but should not care of the underlying cause of the fuzzy bits.
- **Example**: Dungeon master Track 0, sector 7

4.6.2 MFM Timing Violation

- **Description**: These fuzzy bits are obtained by using flux reversals that violate the timing of the MFM rules.
- **Creation**: These fuzzy bits are obtained by placing flux reversals that contains MFM timing **violations** (data separated by less than 4 µs or more than 8 µs). For example a long series of zero data with missing clock bits. These bit-cell width are beyond the normal DPLL capture range and the next received reversal will be interpreted differently based on small random variation of the DPLL clock and/or the drive rotation speed. Of course this technique requires special hardware that has capability to vary the FDC clock on the fly, or the capability to directly control the bit cell width/position (e.g. the Discovery Cartridge). Note that this violations are often achieved by using an unformatted a section of the track. See Unformatted Diskette / Track / Sector section for more information.
- **Detection**: As mentioned this protection results in Fuzzy Sector. Therefore it can be detected by reading the same fuzzy sector (i.e. sector that contains fuzzy bits) several

times and checking that returned data are random. Without specific hardware it is not possible to find the real underlying cause of the fuzzy bits but this information is of no use for an emulator or a duplicator.

- **Duplication**: Difficult and requires special hardware (i.e. the Atari WD1772 cannot be used to copy this kind of bytes). Analog or digital copiers can be used but, as usual, digital copier should be preferred.
- **Emulation**: The preservation file should have an indicator to record the fact that the sector is a fuzzy sector but should not care of the underlying cause of the fuzzy bits.
- **Example**: D50 Editor - Track 0 - Sector 10.

4.6.3 Weak Bit

- **Description**: We use the term ***weak bits*** for data bits that produce **weak flux reversals** below a certain threshold that will therefore result in ambiguous reading returning different values on different reads (see fuzzy bits for a generic description). The SpinRight documentation (from **SpinRite's** Defect Detection Magnetodynamics site) gives a good explanation on weak recorded reversals.
  Weak bits can be created by many different means but the most popular have being described in the US Patent 4,849,836.
  One method consists to move the head slightly out of alignment during write operation (see figure 3).
  As the Atari FD drives do not have a sophisticated track follower mechanism, this result in weak reversals during read (see figure 4).
  Another method consists in writing a “protection track” in between normal tracks (see figure 5). It is obvious that this extra track will induce perturbations in the data bit flux of the adjacent tracks resulting in weak bits when there is opposition in the fluxes.
  Yet another method consists in placing bits on top of *physical defects* on floppy surface. To be useful these defects have to be created precisely on specific spots of the surface layer using for example evaporation with an infrared laser.

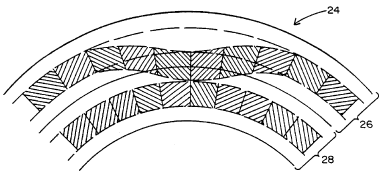


FIG. 3

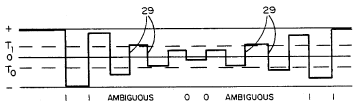


FIG. 4

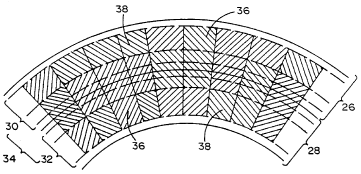


FIG. 5

- **Creation**: Creation of this type of weak bits requires very specialized hardware. Here we are not talking about special floppy disk controllers but about special floppy drives.
- **Detection**: As mentioned this protection results in Fuzzy Sector. Therefore it can be detected by reading the same fuzzy sector (i.e. sector that contains fuzzy bits) several times and checking that returned data are random. Without specific hardware it is not possible to find the real underlying cause of the fuzzy bits but this information is of no use for an emulator or a duplicator.
- **Duplication**: It is obviously at least extremely difficult if not impossible to exactly reproduce the weak bits described in this section. However it is possible to mimic their behavior by placing Flux Reversals in Ambiguous area as this result in the same behavior and therefore should be transparent to the detection mechanism of the protected program.
- **Emulation**: The preservation file should have an indicator to record the fact that the sector is a fuzzy sector but should not care of the underlying cause of the fuzzy bits.
- **Examples**: I am not aware that this technique has been used on Atari.

4.7 Write Splices

4.7.1 Sector write splices

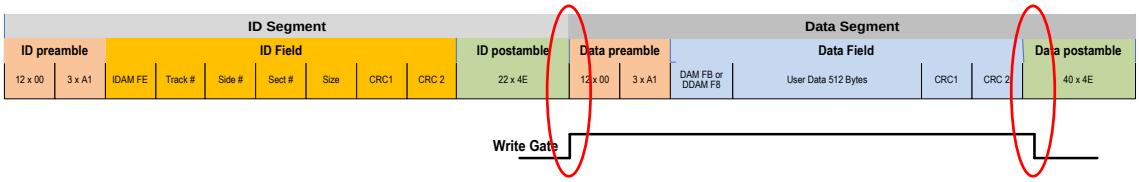
In general it is not possible with the WD1772 FDC to write an entire track in one operation. This is due to the fact that when writing a track the data in the range 0xF5 to 0xF7 are treated as special control bytes and therefore they cannot be written directly during a write track (format) operation. Therefore on an Atari machine, to use a floppy disk, the first operation consists in formatting the track using a ***write track*** command then the data field of each sector is written using ***write sector*** commands.

Here is a simplified description of the write sector command:

Upon receipt of the Write Sector Command the FDC searches on the track an ID field that has the correct track number, correct sector number, and correct CRC. If such an ID field is found the WD1772 counts 22 bytes from the CRC of the ID field and it activates the WG output. Then the following data are written on the disk:

- 12 bytes of zeroes
- Three A1 Sync
- A Data Address Mark
- The complete data field
- A two-byte CRC is computed internally and written on the disk
- One byte of logic ones.

Then the WG output is deactivated.



It is easy to understand that the activation and deactivation of the Write Gate cannot be aligned precisely with the original data written during the format operation and furthermore, as the speed of the drive fluctuates, the overall size of the data segment will most likely not be the same. This results in writing non-aligned (drifted) transitions, which most likely violates MFM rules, when the WG is activated and deactivated. These two areas are called sector write splices.

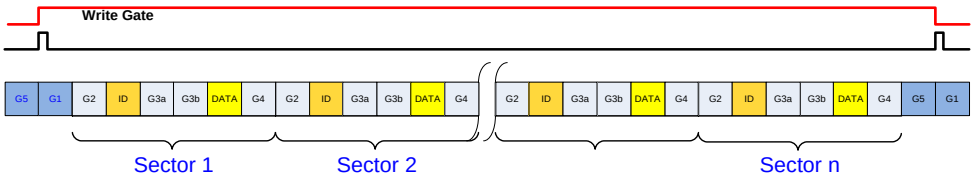
When data are written on a floppy disk with professional duplication equipment the complete track is written in one operation and therefore these sector write splice areas **do not exists**. This is one way to verify that we are working on a “master floppy disk” written on professional duplication equipment and that the sector data on the FD has not been tempered.

4.7.2 Track write splices

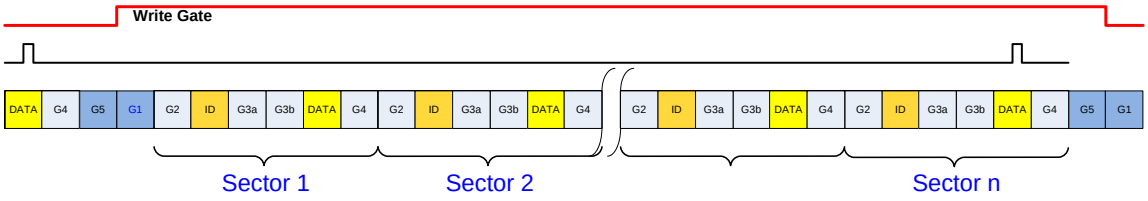
As said before when using professional duplication equipment it is possible to write a complete track in one operation and this will result in a track **without** any sector write splices. The writing of the complete track starts at a specific point of the rotation of the disk and ends up at a specific point (typically the same point). For non-protected tracks these starting and ending points (WG activation / deactivation) are aligned with the track index.

But even with professional equipment it is no possible to deactivate the write gate at precisely the same position that when it was activated. This results in writing a non-aligned (drifted) last transition, which most likely violates MFM rules, when the WG is deactivated. This area is called the track write splice.

In the case of a standard track the writing starts in the post-index gap G1 and stops in the pre-index gap G5 and therefore the track splice happen in this non-critical area.



However some protections are based on shifting the position of the complete track in respect with the index. For example the index might be positioned in the middle of the last data field (data over the index protection).



In such a case it is not possible to activate and deactivate the Write Gate at the position aligned with the index because this would result in a track write splice located in the middle of a data field and this would therefore result in corrupted data.

For this kind of protection it is therefore important for the mastering equipment to activate and deactivate the write gate in a position which is still located close to the border of the pre-index and post-index gaps. However in that case the location of the track write splice can be located anywhere with respect to the index.

4.8 Sync Address Mark

4.8.1 MFM Address Marks reminder

With MFM encoding a clock is only added for two consecutive 0 data bits and it is therefore not possible to directly differentiate between clock and data bits on arbitrary sequence of bits. Therefore the synchronization is done by searching for a sequence of special bytes.

First, there is a special data sequence encoded at the beginning of each ID and DATA field: a long string of zero's. This sequence provides to the DPLL enough time to adjust the frequency and center the inspection window. This is especially important for the DATA field because a Write splice occurs when the read/ write head starts re-writing a data field. The slight variations in the rotational speeds cause the first flux change to occur in different positions for each write operation and therefore the DPLL needs to adjust to this new frequency/position. But this is not part of the synchronization.

Second, special bytes are encoded that violates the MFM encoding rules: either $A1 or $C2 bytes are written, with a missing clock in one of the sequential zero bits. These 2 special bytes with missing clocks are called the Sync Mark Bytes. In practice there is a sequence of 3 consecutive Sync Bytes that should normally be followed by an Address Mark[5] (IAM, IDAM, or DAM) as described in the track format.

In summary if a sequence of zeros followed by a sequence of three Sync Bytes is found, then the PLL (phase locked loop) and data separator are synchronized and data bytes can be read. The following table shows the Sync Bytes and the Address marks used by the WD1772 on the Atari ST

ADDRESS MARK	Data	Clock	Missing clock between bits	Resulting Bit Sequence
Sync Byte	$A1	$0A	4 & 5	0100010010001001 ($4489)
Sync	$C2	$14	3 & 4	0101001000100100 ($5224)
Index Address Mark	$FC			
ID Address Mark	$FE			
Data Address Mark	$FB			
Deleted Data AM	$F8			

The bits order is from MSB to LSB (the way they are sent) with first bit being numbered 0. Note that this encoding ***does not violate the 1,3 RLL rules*** (sequence of 4 consecutive 0) and therefore it is possible to find in a stream some bit configuration that are the same as the $C2 synch mark. This is known as the false sync mark problem/bug during a read track command. Normally the $C2 sync bytes are only used before an IAM and therefore normally not used in standard Atari diskettes.

[5] Note that, in MFM, for the address marks characters between $F8 and $FF the least significant bit is always ignored by the WD1772 and therefore : $F8=$F9, FA=FB, FC=FD, $FE = $FF

4.8.2 Overlapping Sync Mark

Some sequence of overlapping sync mark that are impossible to create on an Atari can be used for protections.

- **Overlapping 4489 4489 (A1-A1)**

We take the 4489 pattern and we try to find how it can overlap a 4489 pattern

```
0100010010001001
        0100010010001001

0100010010001001
               0100010010001001
```

- **Overlapping 5224 4489 (C2-A1)**

We take the 4489 pattern and we try to find how it can overlap a 5224 pattern

```
0101001000100100
     0100010010001001

0101001000100100
            0100010010001001
```

The first sequence happen normally for all ID and DATA records as the 3 sync bytes are preceded by a series of 0 bytes (10 in MFM).

- **Overlapping 4489 5224 (A1-C2)**

We take the 5224 pattern and we try to find how it can overlap a 4489 pattern

```
0100010010001001
               0101001000100100
```

- **Overlapping 5224 5224 (C2-C2)**

If we take two 5224 pattern and we try to find how they can overlap we can see that this is not possible.

- Test Patterns

For test we can take the overlapping sequences described and precede them by a sequence of 00 byte (1010 in MFM) to guaranty a synchronization of the DPLL:

```
1010 1010 1010 1010 1010 1010 1010 1010 1010 0010 0100 0100 1000 1001 = AAAA A244 89AA
1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 0100 0100 1000 1001 = AAAA AA44 89AA
0101 0101 0101 0101 0101 0101 0101 0001 0010 0010 0101 0010 0010 0100 = 5555 5551 2252 2455
```

Chapter 5. Analysis of Games/Programs

This section provides detailed analyses of some programs/games using key disk protection. The analyses have been done with the goal to illustrate the usage of the protections described in this document.

However it must be noted that:

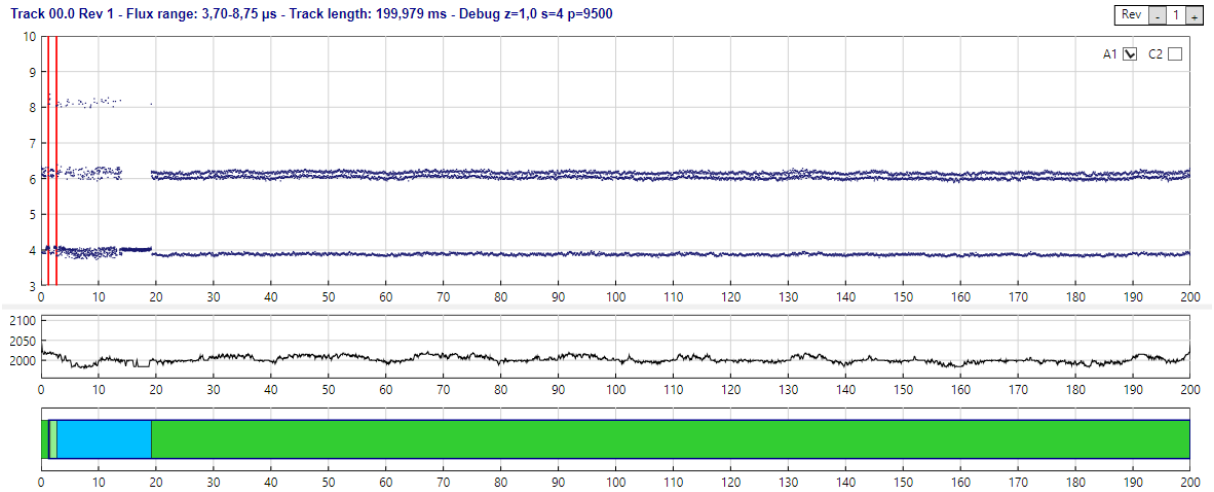
- The presence of a described protection mechanism does not imply that it is actually used.
- It is possible that for a game analyzed in this section, more protections than the one described here exist.
- Beware that diverse releases of the same game may exist and may use different protections.
- Only Original diskettes have been used (unless specifically noted). However it is difficult to know for sure that a diskette has not been modified before the preservation image was created.
- All the floppy diskettes imaged are from the 80's 90's and may be damaged by time or stress from the environment.

5.1 Barbarian (from Psygnosis)

It seems like there are several versions of this game that uses **different** protections. The version I have has the following protections:

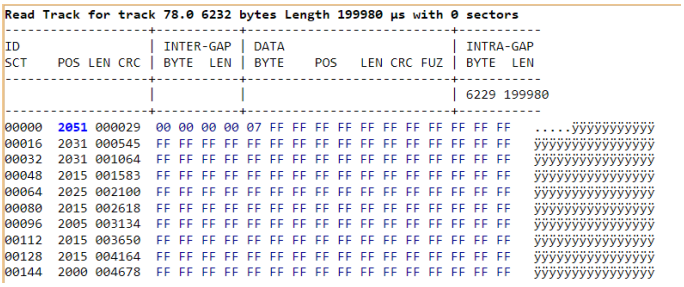
- ✶ Non Standard Number of sector: Track 0 only one sector
- ✶ Track Not Found: Track 74-79

If we look at track 0 of Barbarian we first see that the layout is rather unusual as the track has only **one** sector of 512 bytes! The sectors on other tracks are numbered from 10 to 18.



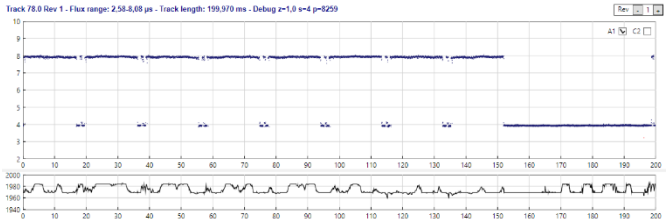
The track 78 seems unformatted (no sync mark, no sector) but however is MFM formatted.

The program performs up to 10 times read track commands on track 78 and tests for value 0xFF or 0x00 at location 28 (0x1C).



This is obtained with a special track that uses a continuous sequence of 4 µs flux transitions. This results in a MFM string like this ...1010101010101010... Depending on which bit the sampling starts this result in values 0x00 or 0xFF. The switch from 0x00 to 0xFF that can be seen at the start of the track is due to the presence of few glitches at the beginning of the track. It is unclear if this is done on purpose.

I have another version of Barbarian that also tests for 0x00 or 0xFF on track 78 but use a more difficult to produce stream of flux transitions. In this version the track uses a sequence of 8µs fluxes. This results in a MFM string like this ...1000100010001000100010001000... Depending on which bit the sampling starts this result in MFM values: 1111 2222 4444 8888 that decodes into 0x55 0x00 0xAA 0x00 respectively.



For this protection the program tests the presence of 0x00 or 0xFF, therefore Pasti imager and Aufit 0.4b+ always decode this kind of sequence as 0x00 in order for the protection to succeed.

5.2 Bob Morane

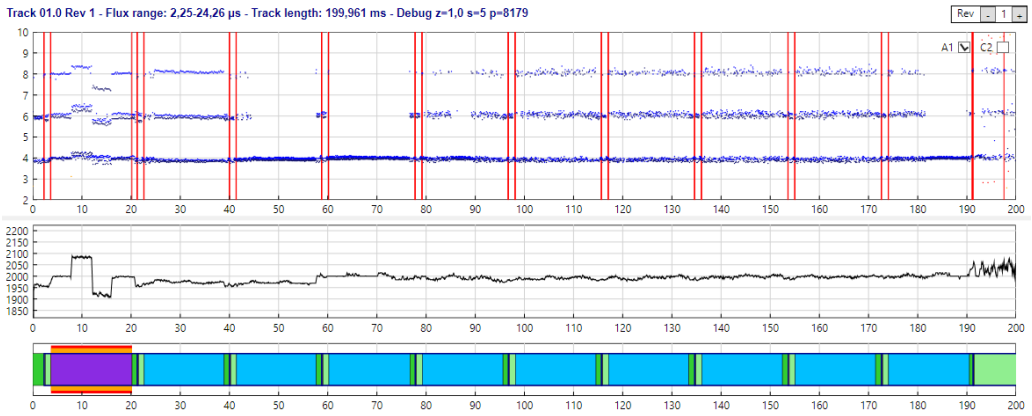
Track 50.0: All the “normal 0x4E” gap byte are replaced by the “invalid 0xF7” byte.

```
00000  1926 000024  BF BF A1 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
00016  1984 000536  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
00032  1979 001046  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
00048  1973 001555  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 00
00064  1984 002066  00 00 00 00 00 00 00 00 00 00 00 00 A1 A1 A1 FE 32
00080  1984 002574  00 01 02 0B EE F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
00096  1988 003082  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 00 00 00 00 00
00112  1994 003594  00 00 00 00 00 00 00 A1 A1 A1 FB 87 EA 00 00 87
00128  1992 004103  EA FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
```

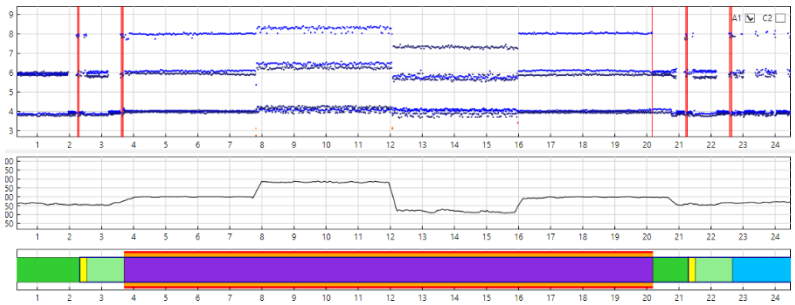
5.3 Colorado

- Track 01.0 I find the following protections:
  - Intra-sector Bit-rate Variation (IBV)
  - Sector with Fuzzy Bits (FZS) and Bad Data (CRC)
  - Invalid ID Field (IIF) without Data Field (SND)

Here is a plot of the complete track:



If we zoom in the first sector we can see some **intra-sector** clock rate variation. If we look at the intra-sector bit-rate variations we recognize a **Macrodos** protection from **Speedlock**.



Here we can see that the data field is roughly dived into four segments. In the first segment we have normal timing, in the second segment we have above normal clock values, in the third segment we have below normal clock values, followed by the last segment with normal values. This corresponds well to the definition of IBV where we have the sector divided into 4 regions with timing: normal, above, below, and normal. Note that each segment is about 128 bytes and that the above and below clock rate compensate. This means that the overall length of this sector is 16487.46 µs which is very close to a normal 16480 µs sector.

Probably due to the quick shifting of the clock we have some border bits and therefore the sector also reads with fuzzy bytes and CRC error.

Now if we read the complete track and look at the end of the buffer we have some strange values:

Here we can see an abnormally long sync sequence followed by an IDAM with the following errors: Nonstandard IDAM (IIF-NSI), Invalid track number (IIF-ITN), Invalid sector length (IIF-ISL) and Invalid ID CRC (IIF-IIC)

```
05952  2000 189141  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
05968  2000 189653  FF FF FF FF FF FF FF FF FF FF F0 7F FF FF 9F FF  ÿÿÿÿÿÿÿÿÿÿð.ÿÿ.ÿ
05984  2013 190166  FF FF FF FF FF FF FF FF FF FF FC F8 02 12 12 12  ÿÿÿÿÿÿÿÿÿÿüø....
06000  2023 190681  12 12 12 12 1F FF FF FF FF FF FF E1 A1 A1 A1 A1  .....ÿÿÿÿÿÿá¡¡¡¡
06016  2048 191181  A1 A1 A1 FF 0E A4 96 A4 02 01 65 18 40 9C 9C 9C  ¡¡¡ÿ.¤.¤..e.@...
06032  2000 191703  9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C  ................
06048  2000 192215  9C 9C 9C 9C 9C 9C 9C 9C 9C 9D FF FF FF FF FF FF  ..........ÿÿÿÿÿÿ
06064  2037 192734  FF FF FF FF FF FF FF FC 39 00 EA 29 5F 2A 0E 54  ÿÿÿÿÿÿÿü9.ê)_*.T
06080  2039 193254  55 55 00 02 F0 01 32 1F 00 01 00 02 00 04 F5 96  UU..ð.2.......õ.
06096  2015 193778  02 BE 80 AC 07 21 04 9D 80 08 10 04 05 18 40 C1  .¾.¬.!........@Á
06112  2048 194295  62 90 00 C1 B1 30 2C 00 22 38 20 E1 E0 2A 00 58  b..Á±0,."8 áà*.X
06128  2007 194810  12 0A 77 94 28 41 02 AE A5 0B 20 00 78 A0 04 00  ..w.(A.®¥. .x ..
```

5.4 Computer Hits Volume 2 (Beau-Jolly)

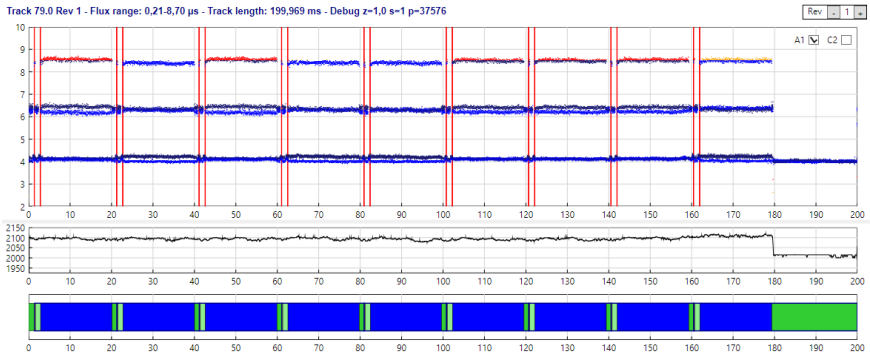
This release is a set of two diskettes that contains a compilation of the following games: Disk 1: Tau Ceti, Tetris – Disk 2: Joe Blade, and Tracker.

Computer Hits Volume 2 uses the following protection mechanisms:

- ✶ Short track 79 of diskettes 1 and 2 (Long Sectors)
- ✶ Nonstandard Sector's Number: 11 Sectors/Track
- ✶ Data Beyond Index pulse on tracks 0-78 of diskette 2

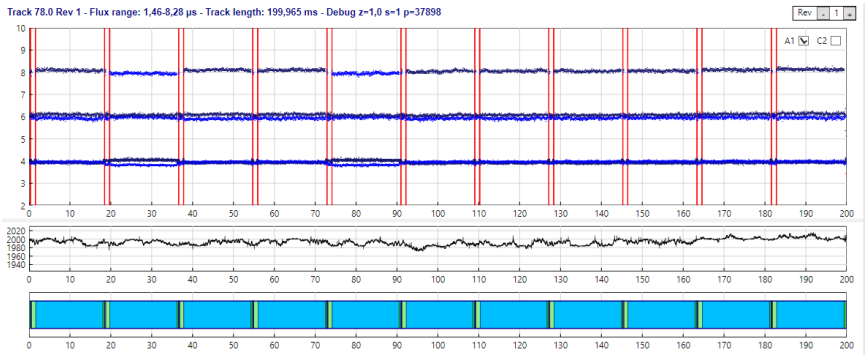
Track 79.0 – Short track

The bit cell width of bytes written in all the sectors of track 79 (disks 1 and 2) is 4.2 µs (5% above normal). After the last byte of the last sector the cell width is returned to the standard 4µs value. This results in a short track with less than 6000 bytes instead of a normal 6240 bytes track.



Track 0-78 – 11 Sector tracks

Using 11 sectors per track is not really a protection, however it pushes the WD1772 at its limit.



For example let's look at the layout of track 0 of the first diskette.

```
----------+-----------------+----------+-------------------------+-----------
GAP2      |ID               |GAP3      |DATA                     |GAP4
Bt   Lgt  |Sct Pos    Lgt CRC|Bt  Lgt  BS|Bt   Lgt   CRC TMV BRD Clk |Bt  Lgt   BS
----------+-----------------+----------+-------------------------+-----------
37   1187 |NO ID            |0   0     0|515  16442 OK  0   0   3.99|4   127    0
11   351  |5   18109  223 OK |31  987   0|515  16447 OK  0   0   3.99|4   127    0
11   350  |9   36246  223 OK |31  987   0|515  16440 OK  0   0   3.99|4   127    0
11   349  |2   54373  202 BAD|32  1002  0|515  16384 BAD 0   0   3.98|5   153    0
11   350  |6   72467  223 OK |31  983   0|515  16406 OK  0   0   3.98|4   127    0
11   351  |10  90559  223 OK |31  989   0|515  16405 OK  0   0   3.98|4   127    0
11   349  |3   108655 222 OK |31  984   0|515  16402 OK  0   0   3.98|4   127    0
11   351  |7   126743 223 OK |31  986   0|515  16428 BAD 0   0   3.99|5   153    0
11   352  |11  144887 224 OK |31  989   0|515  16395 OK  0   0   3.98|4   127    0
11   349  |4   162974 222 OK |31  985   0|515  16437 OK  0   0   3.99|4   127    0
11   350  |8   181097 222 OK |31  984   0|515  16447 OK  0   0   3.99|18  572    0
20   632  |0   199958 19  BAD|0   0     0|NO DATA                  |0   0      0
----------+-----------------+----------+-------------------------+-----------
```

What we see is that the layout of this track uses strange values for the number of bytes in GAPS. The GAP3 is set to 31 bytes and GAP4 is only 4 byte. Normally Gap3 must be 37 bytes (22x$4E+12x$00+3x$A1) and is not compressible. This corresponds to the time it takes for the WD1772 to switch from reading an *ID Field* to a *Data Field.* Here we have the following GAP3

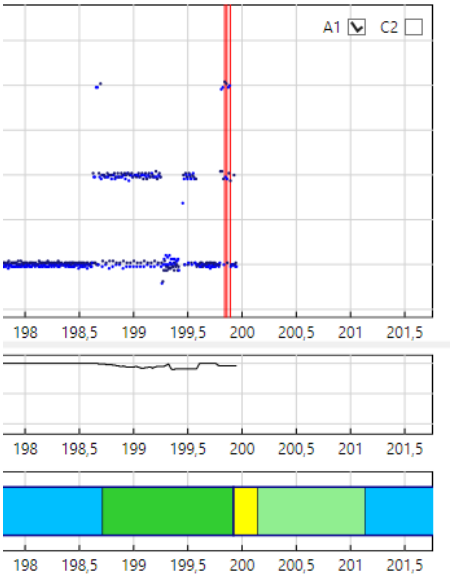
```
+ GAP3 31 bytes @18333 us length=987.49 us - TMV=0 BRD=0 BS=0 IDG=0
  023e 18333  4000  4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e  NNNNNNNNNNNNNNNN
  024e 18844  4000  4e 4e 4e 4e 4e 4e 00 00 00 00 00 00 a1 a1 a1     NNNNNN.........
```

This format results in a **"read only" track** because we have the normal 22x$4E bytes (GAP3a) but we only have 6x$00 bytes. If a write is done on such sector the write gate is

raised at the end of the ID postamble (GAP3a) then the WD1772 write 12x$00 3 sync bytes and a normal data field. This results in the sector to be shifted by 6 bytes but the data postamble (GAP4) is only 4 bytes and therefore we are already in the next sector!

Track 0-78 (disk 2) – Data beyond Index

The last sector of each track is pushed very close to the end of the track. If you look at the diagram you will see that in fact the sync mark and the first byte of the ID field are located before the index and the rest beyond the index including of course the data field. It is absolutely impossible to do this on an Atari and this requires very precise mastering machine. As the ID field is located "above" the index it is not possible to start/stop the writing aligned with the index (this would result in write splice inside the ID field).



If you look carefully at the transitions between 199 and 199.5 it seems that we have the track write splice at this location. This means that the mastering machine would start / stop writing about half millisecond before the index.

If we look at the end of this track buffer we find something like:

```
+ GAP2 20 bytes @199325 us length=632.73 us - TMV=0 BRD=0
  186d 199325 4000  ff ff ff ff fe 01 39 39 39 38 00 00 00 00 00 00  ......9998......
  187d 199840 4000  02 a1 a1 a1                                      ....
= ID=0 1 bytes @199958 length=19.95 T=0 H=0 S=0 Z=512 CRC=0000 *** BAD *** TMV=0 BRD=0 BS=0
  1881 199958 4031  fe
```

As you can see here we only have a sync sequence followed by an IDAM but not the rest of the ID field (remember the read track command terminates at the index). This start of the ID field (the IDAM) is therefore at the **very end** (only few micro seconds) of the track and therefore the rest of ID field must be at beginning of track.

Therefore if you do a ***read track*** command on a real Atari you have all the chance not to see this ID field. For example here is the content of the end of the track buffer as read by the ***Panzer*** program on a real Atari:

```
1830 3973  ff 80 00 00 00 3f ff ff ff 80 00 00 00 3f ff ff  .....?.......?..
1840 3973  ff 80 00 00 00 3f ff ff ff 80 00 00 00 3f ff ff  .....?.......?..
1850 4037  ff 80 00 00 00 3f ff ff ff e0 10 c8 48 48 48 48  .....?......HHHH
1860 3973  48 48 48 48 48 48 48 48 00 00 00 00 00 00 00 00  HHHHHHHH........
1870 4069  00 00 10 90 90 90 90 ff ff ff ff ff ff ff c2 a1  ................
```

Here you can see that we have the start of the sync sequence but not the IDAM. This is due to the Atari DMA circuit: the DMA always delivers multiples of 16 bytes due to the buffering mechanism and therefore up to 15 bytes may be "stuck" in the DMA buffer at the end of the ***read-track*** command.

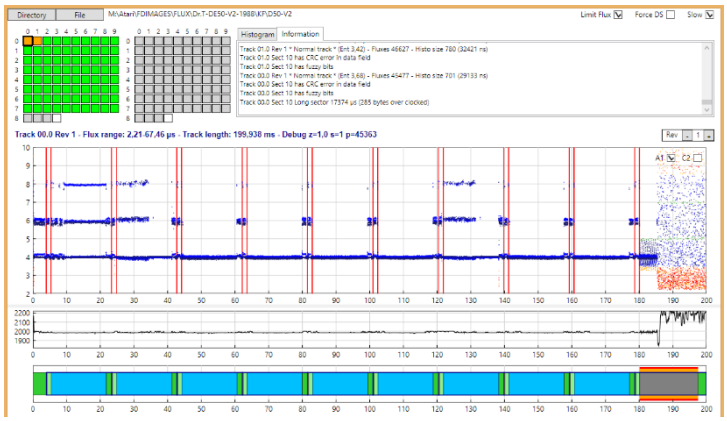
However the WD1772 will detect this ID field without problem with a ***read-address*** command and will find the corresponding DATA field with the ***read-sector*** command.

Therefore it looks almost impossible to position this *ID Field* with this precision by software and mastering machines are required.

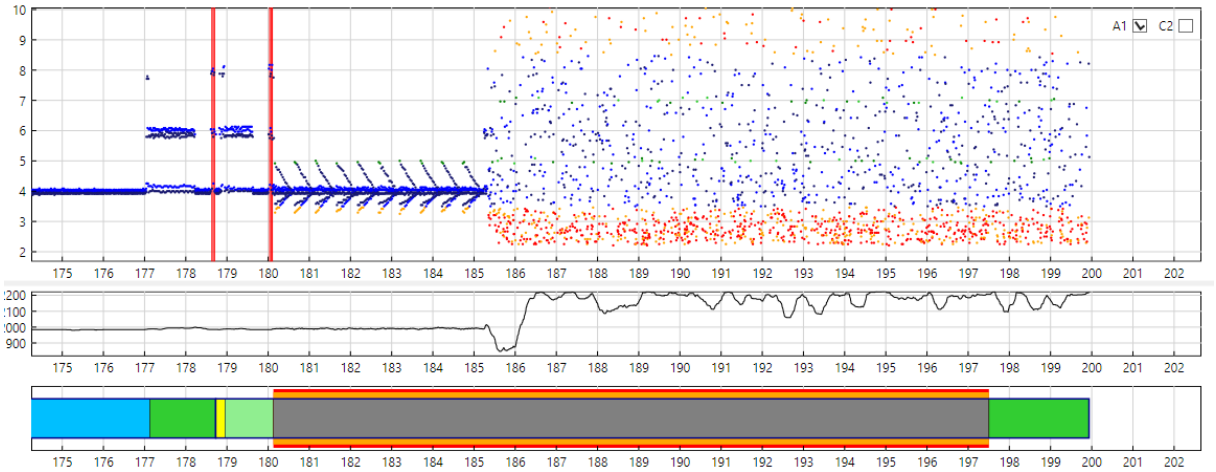
5.5 D50 Editor V2 (DrT)

The D50 Sound Module Editor program from DrT uses the following protection mechanisms:

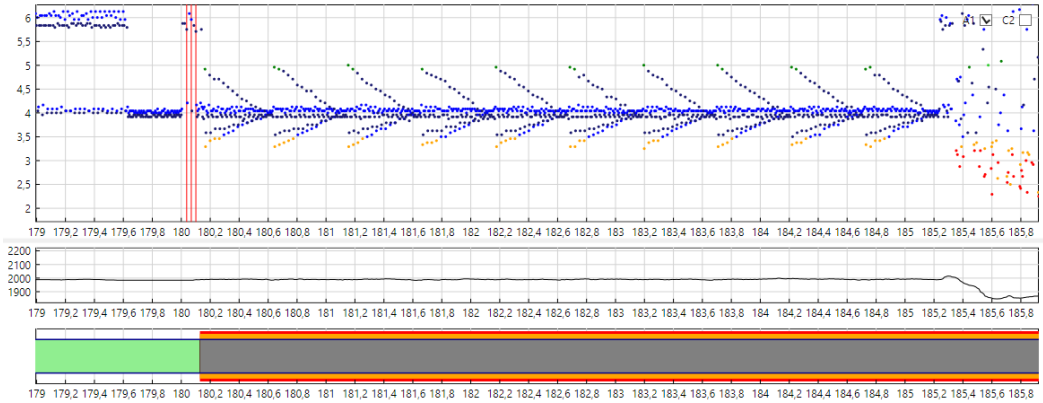
- ✶ Track 00.0-79.0 sector 10 has fuzzy bits in the *Data Field*.



We can see that sector 10 is partially unformatted but it has also a lot of border bits. Here is a zoom on sector 10:



If we further zoom we can see that after the position 180000 we have a lot of border bits in the range from 3 to 5µs with a strange pattern that “compensate in pair”. This results in an average 2000 µs cell but for sure all these border bits should generate fuzzy bits.



After the position 185000 we can see that we have random flux reversals. This pattern is typical of an unformatted track. Therefore we can conclude that the formatting of the track is stopped after about one third of the last sector. This is obviously not feasible with the WD1772 FDC and therefore to copy this track it is necessary to have special mastering device like Discovery Cartridge or KryoFlux board or Supercard Pro.

Note that random flux reversals result into unpredictable clock frequency (and also unpredictable inspection windows position) of the DPLL. This and the presence of border bits results in fuzzy bytes in the sector.

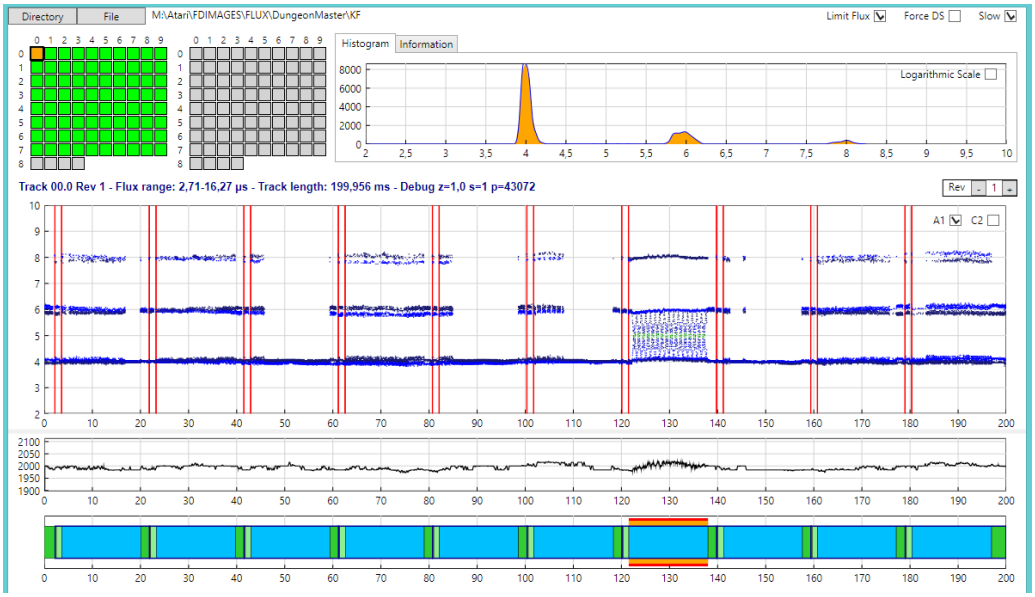
5.6 Dungeon Master (FTL Inc.)

For detail analysis of the Dungeon Master & Lost Scroll protection please refer to the DM Protection document, the detailed analysis of the Dungeon Master and Chaos Strikes Back for Atari ST Floppy Disks and the US patent "Copy Protection for computer Disc 4,849,836")

The game "Dungeon Master" uses the following protection mechanisms:

- Invalid Sector Number: Track 0 the sector 8 is numbered 247.
- Fuzzy bits & Sector with bad *Data*: Track 0 sector 7 the *Data Field* has bits in Ambiguous areas resulting in a fuzzy sector with CRC error.

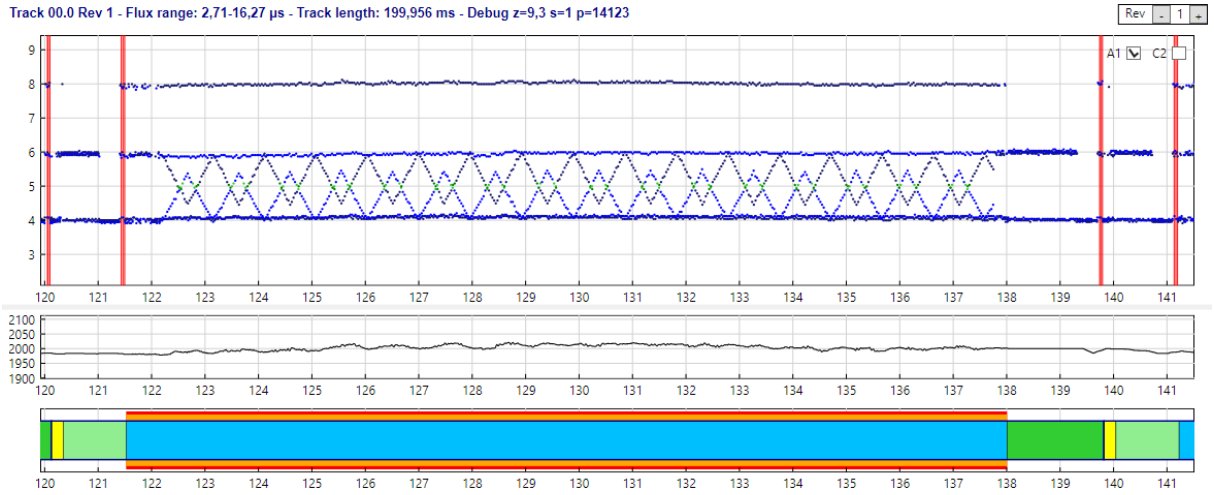
Here is the Layout of track 0



As you can see in sector 7 we have a lot of **border bits** (BRD) aka bits in Ambiguous area. Looking at the content of this sector we can see that the clock period range from 3938 ns to 4031 ns with an overall clock period of 4.01 µs

```
Detail buffer content for sector 7 with 515 bytes
= DATA ID=7 515 bytes @121545 us length=16506.79 CRC BAD CLK=4.01 TMV=0 BRD=495 DOI=0
  *** Fuzzy Sector *** starting at byte position 34
  0000 121545 3968  fb 07 50 41 43 45 2f 46 42 09 53 65 72 69 ca 08  ..PACE/FB.Seri..
  0010 122055 3968  00 00 ef e9 01 68 68 68 68 68 68 68 68 68 68 68  .....hhhhhhhhhh
  0020 122565 3938  68 68 68 e8 e8 e8 e8 e8 e8 68 68 68 68 68 68 68  hhh......hhhhhhh
  0030 123073 3968  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  0040 123583 4031  68 e8 e8 e8 e8 e8 68 68 68 68 68 68 68 68 68 68  h......hhhhhhhhh
  0050 124092 3968  68 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 e8  hhhhhhhhhhhhh..
  0060 124604 4000  e8 e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68  ....hhhhhhhhhhh
  0070 125114 4000  68 68 68 68 68 68 68 68 68 68 68 68 e8 e8 e8 e8  hhhhhhhhhhh....
  0080 125628 3968  e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 68  ...hhhhhhhhhhh
  0090 126141 4000  68 68 68 68 68 68 68 68 68 68 e8 68 e8 e8 e8 68  hhhhhhhhh.h...h
  00a0 126654 4031  e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 68 68  ..hhhhhhhhhhhh
  00b0 127168 4031  68 68 68 68 68 68 68 68 e8 e8 e8 e8 68 68 68 68  hhhhhhh.....hhh
  00c0 127683 3968  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  00d0 128197 3938  68 68 68 68 68 68 68 e8 e8 e8 e8 28 68 68 68 68  hhhhhh.....(hhh
  00e0 128710 4000  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  00f0 129226 4063  68 68 68 68 68 68 e8 e8 e8 e8 68 68 68 68 68 68  hhhhh....hhhhhh
  0100 129741 4031  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  0110 130257 4162  68 68 e8 e8 e8 e8 e8 68 68 68 68 68 68 68 68 68  hh.....hhhhhhhh
  0120 130771 4000  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 e8  hhhhhhhhhhhhhh.
  0130 131288 3938  68 e8 e8 e8 e8 e8 e8 68 68 68 68 68 68 68 68 68  h......hhhhhhhh
  0140 131802 4000  68 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 e8  hhhhhhhhhhhh..h
  0150 132319 4063  e8 e8 e8 68 e8 68 68 68 68 68 68 68 68 68 68 68  ...h.hhhhhhhhh
  0160 132831 4000  68 68 68 68 68 68 68 68 68 68 68 68 68 e8 e8 e8  hhhhhhhhhhh....
  0170 133346 3938  e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 68  ...hhhhhhhhhhh
  0180 133858 4031  68 68 68 68 68 68 68 68 e8 68 e8 e8 e8 e8 e8 68  hhhhhhhh.h.....h
  0190 134371 4063  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  01a0 134882 4000  68 68 68 68 68 68 68 e8 68 68 e8 e8 e8 e8 68 68  hhhhhh.hh.....hh
  01b0 135395 4063  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  01c0 135906 4063  68 68 68 68 68 68 68 68 e8 e8 e8 e8 68 e8 68 68  hhhhhhh....h..hh
  01d0 136418 3968  68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68  hhhhhhhhhhhhhhh
  01e0 136931 4000  68 68 68 68 e8 68 e8 e8 e8 e8 68 68 68 68 68 68  hhhh.h....hhhhhh
  01f0 137443 4000  68 68 68 68 68 68 68 68 68 68 68 68 68 68 ac 46  hhhhhhhhhhhhh.F
  0200 137956 4000  42 3a f8                                        B:.
```

Let's zoom to sector 7:



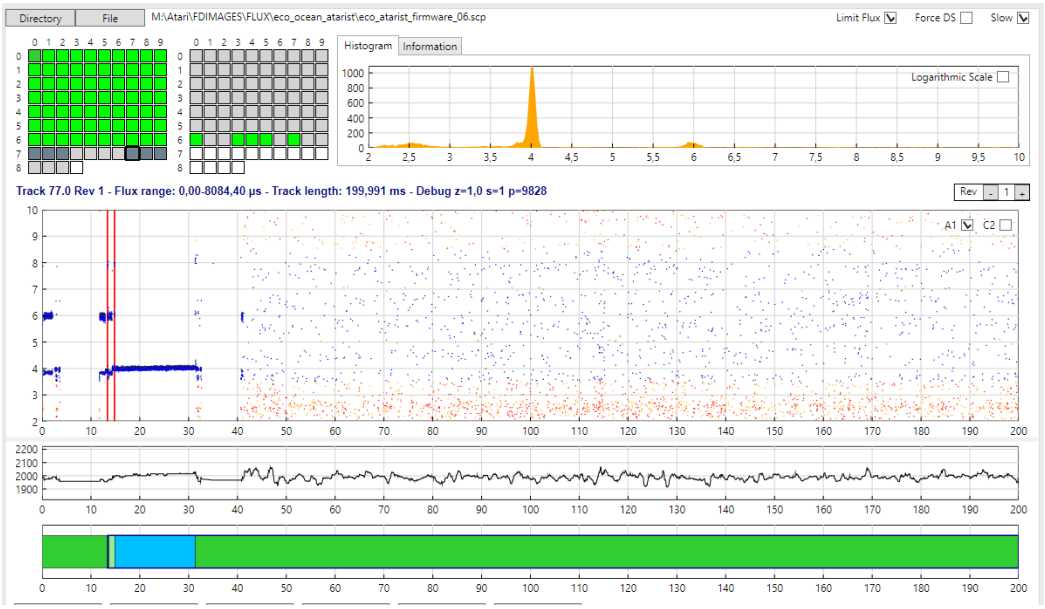
We can see that the flux reversals spacing follow a strange pattern and includes a lot of "border bits".

Here we can see that the beginning of the sector has normal timing. But after the position 122000 we have the bit reversals gradually sliding to the border of the inspection window (close to 5000 ns). We can see that we have a pattern that looks like a sine wave and this implies that many bits are at the border of the inspection window.

As explained in the [WD1772 DPLL Input Circuitry], having reversals at the border of the inspection windows will result in random value latched by the DPLL data separator and therefore these bits can be considered as **Fuzzy Bits**. Reading this sector several times will results in different values returned due to the floppy disk rotation speed fluctuations.

5.7 Eco by Ocean

Tracks 77 and 79 have one sector number 2 and the rest is unformatted. The track look like this

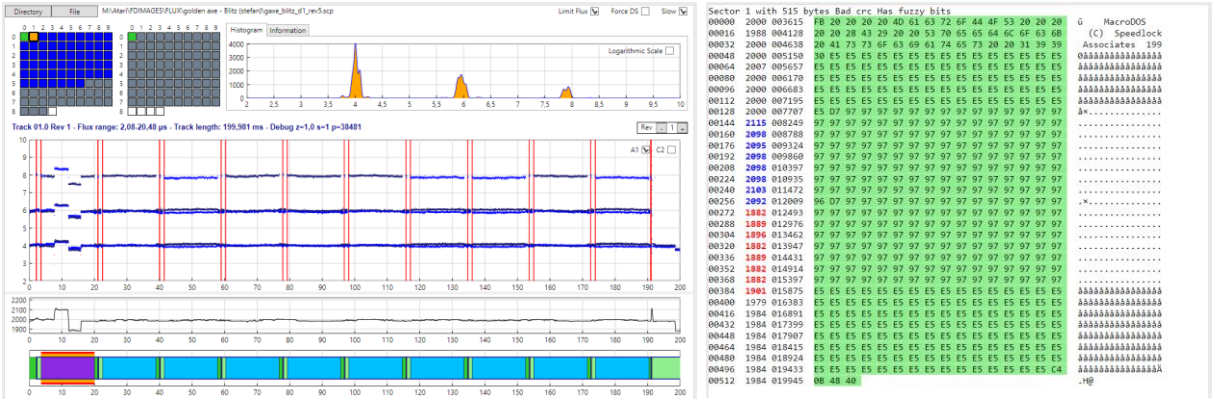


Protection checked using simple read and write calls. Checks that no sector #1 is present on track and then it tries to write on sector #2. If any of this test fails the program freezes.

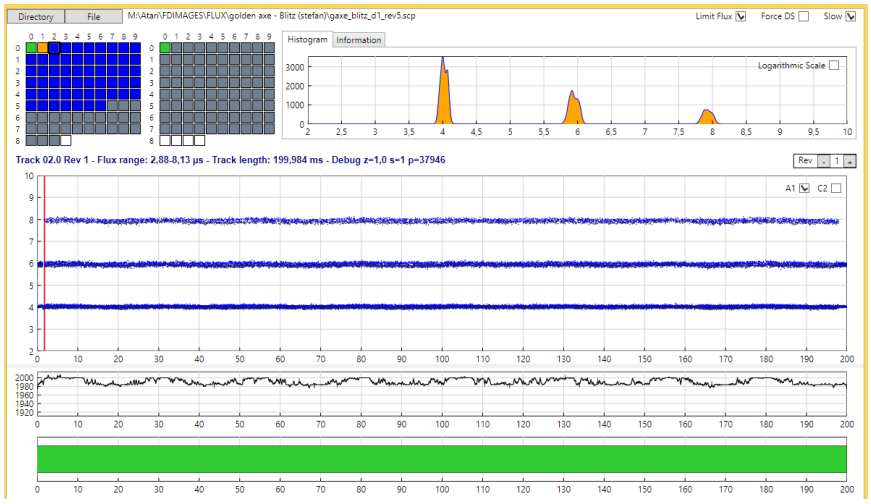
5.8 Golden Axe

On track 01.0 I find the following protections:

- ✶ Intra-sector Bit-rate Variation (IBV) – Macrodos/Speedlock
- ✶ Sector with Fuzzy Bits (FZD) and Bad Data (CRC)
- ✶ Invalid ID Field (IIF) without Data Field (SND) (see Colorado for the Invalid ID field)



Track 02.0 – 56.0 contain Data Tracks.



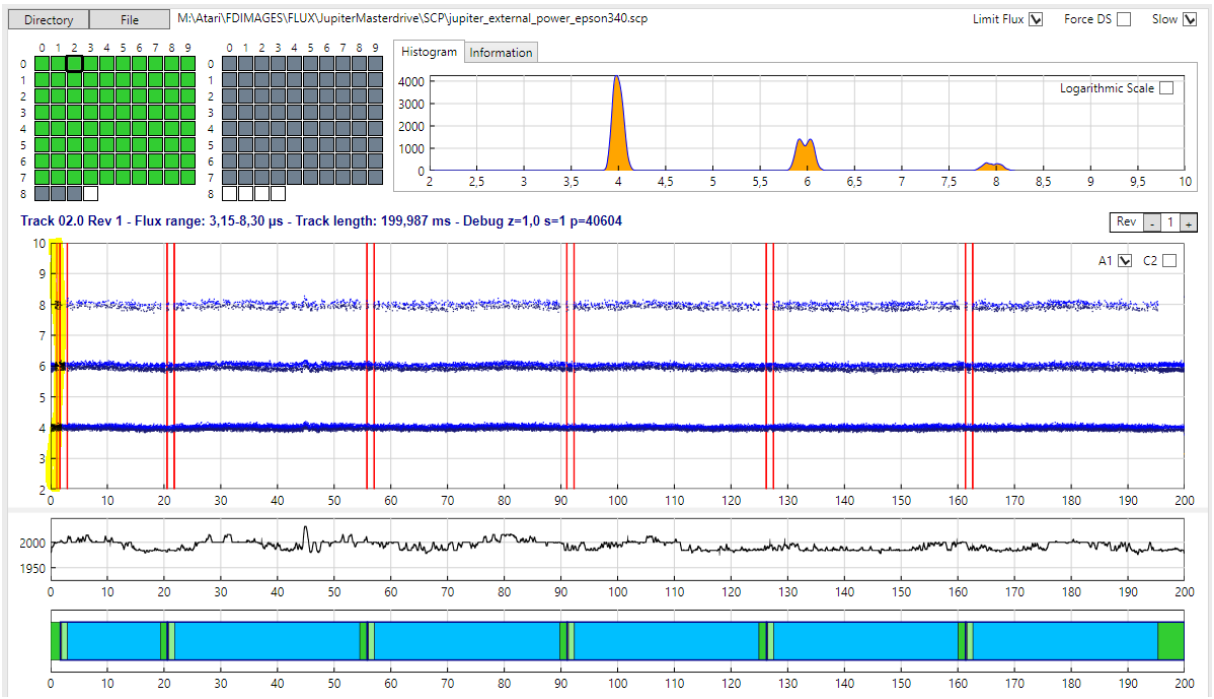
All these tracks are read using a read track command. In this specific version we have a sequence of 3 A1 sync mark followed by the data track. The escape character used is 0x0F.

```
Read Track for track 02.0 6273 bytes Length 199982 µs with 0 sectors
------------------+-----------+---------------------------+-----------
ID                | INTER-GAP | DATA                      | INTRA-GAP
SCT    POS LEN CRC | BYTE  LEN | BYTE    POS    LEN CRC FUZ | BYTE  LEN
------------------+-----------+---------------------------+-----------
                  |           |                           | 6270 199982
------------------+-----------+---------------------------+-----------
00000  1971 000025  02 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E   .NNNNNNNNNNNNNN
00016  1992 000567  4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E   NNNNNNNNNNNNNNN
00032  1992 001077  4E 4E 4E 4E 4E 4E 4E 4E 4E 00 00 00 00 00 00 00   NNNNNNNN........
00048  1988 001590  00 00 00 00 A1 A1 A1 0F 02 0F 00 0F F9 1E 67 85   ....¡¡¡.....ù.g.
00064  2000 002099  1E 21 BE 46 DD 9F 80 47 AD EC 5C F4 0F AC 7A 09   .!¾FÝ..Gì\ô.¬z.
00080  2004 002611  7F 98 5C 70 5F 3A 50 70 00 C2 21 72 8F E6 64 2E   ..\p_:Pp.Â!r.æd.
00096  2000 003124  4E 06 A9 23 5D D4 D7 5B 8B D7 C3 42 2A DC 0F 09   N.©#]Ô×[.×ÃB*Ü..
00112  1990 003635  5B 1A FF 21 46 70 A0 D2 7F A9 3F 77 80 61 8B 0F   [.ÿ!Fp Ò.©?w.a..
00128  2000 004144  09 0F 0A 0F 93 61 24 B9 EA F3 17 E1 BE 63 A1 F8   .....a$¹êó.á¾c¡ø
00144  2000 004658  1A C3 5E A4 BA C2 44 4F FA B8 90 7B F8 55 5E 1D   .Ã^¤ºÂDOú.¸{øU^.
00160  2000 005171  BC 57 37 3F A7 E6 90 8D 30 39 84 A9 39 B9 0F 08   ¼W7?§æ..09.©9¹..
00176  2015 005684  68 64 3C 72 C8 E8 04 2B DE 57 67 8C 6A A7 0F 08   hd<rÈè.+ÞWg.j§..
00192  2000 006192  21 CA 47 DA 64 FF 34 F0 85 FB 7F 4D EB 23 A7 41   !ÊGÚdÿ4ð.û.Më#§A
00208  2000 006709  0C CA B7 C4 F2 B3 0B 84 A2 E5 24 D8 8D BE 85 0F   .Ê·Äò³..¢å$Ø.¾..
00224  2004 007221  D4 E1 8A 0F BF BA A3 28 3E C2 79 57 FC 10 A0 B5   Ôá..¿º£(>ÂyWü. µ
00240  2000 007732  4C 02 41 87 D4 D7 63 E5 E7 4F A6 CD 2E 50 36 16   L.A.Ô×cåçO¦Í.P6.
00256  2000 008244  80 EB 17 43 0E C7 62 31 64 28 EF 6C DC B3 83 27   .ë.C.Çb1d(ïlÜ³.'
00272  2000 008749  DD 12 73 A3 D3 E3 2F 70 30 58 87 E7 E7 48 51 70   Ý.s£Óã/p0X.ççHQp
00288  2000 009261  3D BB 12 63 B3 34 44 BC 35 44 7E A0 D2 81 9C 15   =».c³4D¼5D~ Ò...
00304  2000 009779  A8 B8 79 77 2A 01 BF B2 B9 A6 F9 C0 62 A5 D4 8A   ¨¸yw*.¿²¹¦ùÀb¥Ô.
00320  2000 010289  0F 82 09 C6 25 5D 36 CA 4C 9C 57 B2 F2 9D 8F 45   ...Æ%]6ÊL.W²ò..E
```

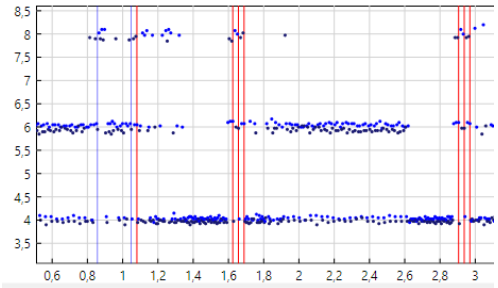
5.9 Jupiter Masterdrive

The protections are located on track 02.0 – 04.0:

- Hidden data into gap
- Invalid data into gap
- Hard to reproduce flux sequence.
- Track size



The game starts by checking if the length of the track is less than 6300 bytes. This is unusual as the normal track length is 6240 but you will see bellow that an extra sequence is added at beginning of the track and this sequence should not add more than 60 bytes. The real hard to reproduce protection is the following: a read track command is done and the content of the buffer is analyzed. The program looks for the first apparition of a $F7 byte (invalid character that can't be written by the WD1772 write track) that must be followed by a $00 byte. If this succeed then it checks that the following bytes contains $DEADC0DE (almost dead code!). Then program checks bytes located before the $F7 for the sequence 921090C20BCDB4**F7**. This is done by comparing the two long word $C2901092 & $F7B4CD0B. So the complete sequence from the first sync mark is:
…C2001C921090C20BCDB4**F7**00DEADC0DE…



This sequence is always read correctly because it starts with a $C2 sync mark. What is also interesting is that the $5224 sync mark ($C2) placed after the $90 is a followed by a $4489 sync mark delayed by one bit (see flux seq. below). You can see on the graph the first $C2 sync mark followed by the $C2+$A1 sync marks part of the protection and latter the normal sequence of 3 $A1 sync marks part of the first ID field. This is an impossible to generate sequence of bit on an Atari and therefore it requires a mastering machine

The two sync marks are decoded by the WD1772 as $C2 $0B:

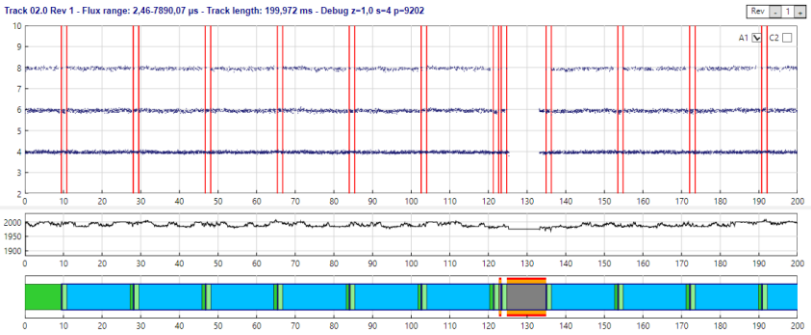
```
Flux Transition Sequence:  9254  A449  4489  5251
1001001001010100101001001001001010010010010001001010010010101010001
C2                0101001010010100
A1                                  0100010010001001
    DSR from C2 to 0B : 11000010, 10000100, 00001000, 00010000, 00100000, 01000001, 10000010, 00000101, 00001011
```

5.10 Kick Off 2 (Anco Software 1990)

Kick Off 2 uses combinations of the following protection tracks 02.0-06.0:

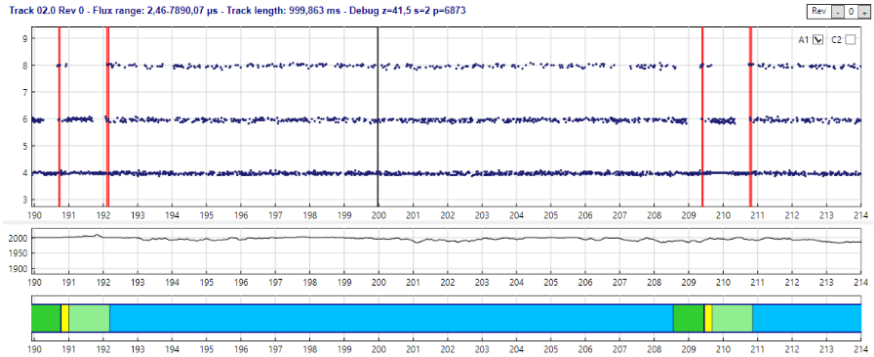
- Nonstandard Sector's Number: 12 Sectors/Track
- Data Over Index pulse
- Sector Within Sector (and even Sector Within Sector Within Sector)
- Non Standard sector Size (1024)
- No Flux Area
- Fuzzy bytes

Here is the complete content of track 2

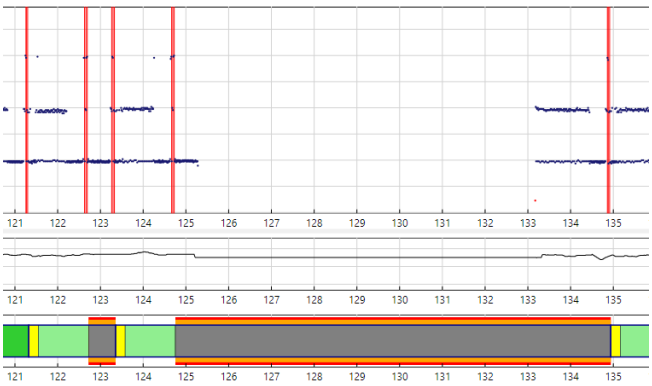


Few things to note:

- We clearly see that we have a No Flux Area (NFA) around 125ms
- Just before this NFA we have several overlapping sectors.
- The last sector continue past the index



If we zoom around the end of track (@ 200ms) we can see that the last sector has its ID field around 191ms and the Data field starts around 192ms and terminate at about 9ms in the next revolution.



Now if we zoom close to the NFA we can see a first sector (sector 0 with a size of 1024) and inside this sector we have a second sector (sector 16 with a size of 1024). This is the Sector within Sector protection (SWS). Both of these sectors included in their data field the NFA area (that reads with Fuzzy bits and CRC error). During mastering the flux transitions are carefully crafted so that the included sector 16 is shifted by a half cell (using a normal $A1 sync mark) from the sector 0. Therefore the read sector command for sector 0 reads the "data bits" of the NFA (remember that during read sector the sync mark detector is disabled). The read sector command for sector 16 reads the "clock bits" of the NFA because this sector is shifted by a half cell compared to sector 0. This technique allows to check the presence of an NFA areas (both clock and data bits equal to zero) in spite of the limitation of the WD1772 that can normally only read the data bits of a sector.

See also Checking NFA with the WD1772

5.11 Maupiti Island

Track 00.0 of disk 1 and disk 2 are normal Atari track with 9 sectors. Track 00.0 on disk 1 contains a boot loader used to load the game. Track 00.0 on disk 2 is used to save game state. All the other tracks on both disk use a non-standard Data track format.

This format contains three standard $4489 sync marks followed by

- $FE
- $07 (escape character),
- track number
- $07,
- checksum high byte
- $07
- checksum low byte
- $07
- 5842 data bytes

The code just searches for two $A1 at the beginning of the track, followed by $FE. It also tests the track number (it has to match the position of the head, bit 7 is set for side 2). The checksum is an unsigned word, which is calculated by adding all 5842 (unsigned) data bytes together. The checksum is also tested.

Several things to note:

- Because the data bytes can contain any byte $00-$FF it can't be written with a standard FDC write track, which can't write bytes $F5-$F7.
- During a read track the sync detector of the FDC is active at all time. It is well known that specific sequence of bytes can be interpreted by the FDC as $5224 sync mark. This causes all the following bytes to be read incorrectly. To prevent that happening a very simple yet efficient method is used: a $07 escape byte is inserted in front any byte that would lead to a sync mark. Claus Brod wrote a wonderful article about this. This is also the reason to have these bytes in the track header! (See also Obitus).

5.12 Night Shift (US Gold)

- Track 0-78
  - 2 Sector 66 Sector with No Data
  - Sector 66 Duplicate Sector

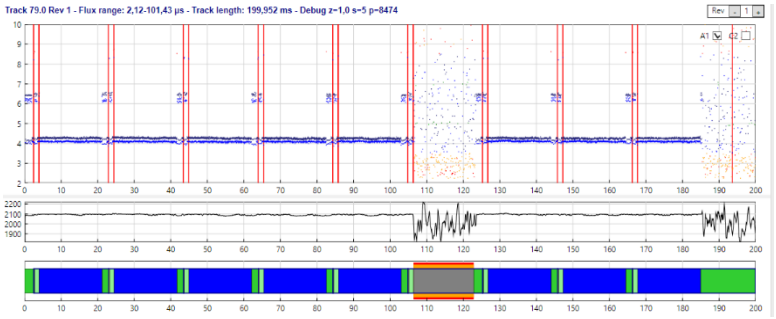
At the beginning of the track we can see that we have a sector 66 not followed by a data segment. At the end of the track we have a duplicate sector 66 also without a data segment. A read sector returns a RNF status for sector 66.

- Track 79
  - Sector 6 Fuzzy bits + CRC error.
  - Short Track (about 6000 bytes)

We can see that sector 6 contain an unformatted area. This result in random byte read (Fuzzy bytes) and of course a CRC error.

It seems that there is another unformatted area at the end of the track.

```
Read Track for track 00.0 6279 bytes Length 199943 µs with 11 sectors
-----------------+-----------+-------------------------+-----------
ID               | INTER-GAP | DATA                    | INTRA-GAP
SCT    POS LEN CRC | BYTE  LEN | BYTE    POS   LEN CRC FUZ | BYTE  LEN
-----------------+-----------+-------------------------+-----------
                 |           |                         |  513 16442
 66  16442 222  OK |    35 1197 |                         |
  1  17842 222  OK |    34 1177 |  515  19252 16381  OK  No |   50  1177
  2  37308 222  OK |    35 1197 |  515  38718 16358  OK  No |   50  1197
  3  56744 221  OK |    34 1175 |  515  58150 16350  OK  No |   50  1175
  4  76171 222  OK |    34 1175 |  515  77577 16373  OK  No |   50  1175
  5  95621 222  OK |    34 1175 |  515  97028 16408  OK  No |   50  1175
  6 115111 222  OK |    34 1177 |  515 116519 16419  OK  No |   50  1177
  7 134618 222  OK |    34 1178 |  515 136028 16463  OK  No |   50  1178
  8 154174 223  OK |    34 1179 |  515 155585 16447  OK  No |   50  1179
  9 173720 223  OK |    34 1181 |  515 175134 16469  OK  No |   50  1181
 66 193288 222  OK |   716 22894 |                         |
-----------------+-----------+-------------------------+-----------
00496   1984 015839  4E 4E 4E 4E 00 00 00 00 00 00 00 00 00 00 00 00
00512   1984 016345  A1 A1 A1 FE 00 00 42 02 C2 40 79 09 09 09 09 09
00528   1984 016836  09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09
00544   1984 017346  0F FF FF FF FF FF FF FF FF FF FF FF F0 A1 A1 A1
00560   1984 017842  FE 00 00 01 02 CA 6F 4E 4E 4E 4E 4E 4E 4E 4E 4E
00576   1994 018351  4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 00 00 00
00592   1984 018863  00 00 00 00 00 00 00 00 00 A1 A1 A1 FB 00 00 4E
00608   1984 019370  4E 4E 4E 4E 4E 57 CB A5 00 02 02 01 00 02 70 00
00624   1984 019880  D0 02 F8 05 00 09 00 01 00 00 00 4E 4E 4E 4E 4E
```



5.13 Obitus

Very similar to Maupiti Island

Track 00.0 is normal Atari track with 9 sectors. All the other tracks on both disk use a non-standard Data track format. However the format of the Data Track is different.

I only know that this format contains the three standard $4489 sync marks and that an escape byte $0F is used to avoid false sync sequence. Apparently most of the track seems used and therefore should allow about 30% more bytes that on a standard track.

```
00000   1922 000025   27 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E   'NNNNNNNNNNNNNNN
00016   1984 000538   4E 4E A1 A1 A1 0F 00 00 4B 58 6F 28 80 64 B7 A2   NN¡¡¡...KXo(.d·¢
00032   1992 001050   84 14 0A BB 0F 0A 0A 01 C7 F0 01 F0 84 5A 23 B7   ...»....Çð.ð.Z#·
00048   1984 001558   FF E8 08 70 1F E0 FF F0 F0 0F F0 0A 28 0F 0A 0F   ÿè.p.àÿðð.ð.(...
00064   2000 002068   A9 A8 3D 45 AA 0F 08 56 6A 9D D4 9A AB 75 46 A8   ©¨=Eª..Vj.Ô.«uF¨
00080   2000 002579   DD 41 AA D7 57 AA 95 D4 EA A9 75 5A A0 C1 04 FF   ÝAª×Wª.Ôê©uZ Á.ÿ
00096   2000 003098   40 F3 00 01 03 FF 83 91 82 80 18 87 15 50 20 10   @ó...ÿ.......P .
00112   2000 003608   33 CC 00 3C 13 F8 10 10 58 40 27 18 AB 44 CA C0   3Ì.<.ø..X@'.«DÊÀ
00128   1997 004122   14 0C C6 B0 00 58 C0 01 3F 01 00 20 80 D8 2D 47   ..Æ°.XÀ.?.. .Ø-G
00144   1986 004631   AF 40 21 AA 46 8F B9 FF 01 20 28 E0 7F E0 A0 E3   ¯@!ªF.¹ÿ. (à.à ã
```

5.14 Operation Neptune

Same as Bob Morane

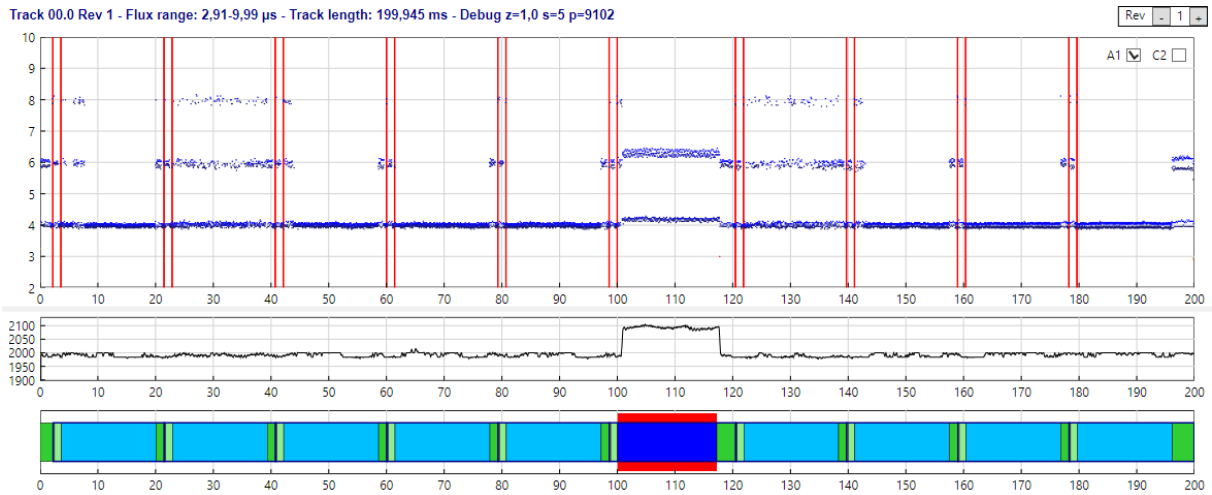
5.15 Populous (Electronic Arts)

Track 00.0

- ✶ Sector 6 has CRC error.
- ✶ Sector 6 long Sector 17206µs which is about 4.2% above a normal sector of 16502µs.

ID				INTER-GAP		DATA					INTRA-GAP	
SCT	POS	LEN	CRC	BYTE	LEN	BYTE	POS	LEN	CRC	FUZ	BYTE	LEN
											67	2249
1	2249	223	OK	34	1180	515	3659	16426	OK	No	43	1180
2	21512	222	OK	35	1198	515	22919	16439	OK	No	42	1198
3	40787	223	OK	34	1179	515	42196	16446	OK	No	42	1179
4	60072	223	OK	34	1177	515	61478	16449	OK	No	42	1177
5	79362	223	OK	34	1180	515	80771	16436	OK	No	42	1180
6	98638	223	OK	34	1180	515	100047	17208	Bad	No	99	1180
7	120527	222	OK	34	1175	515	121931	16398	OK	No	42	1175
8	139757	222	OK	34	1176	515	141162	16428	OK	No	42	1176
9	159023	223	OK	34	1178	515	160431	16439	OK	No	42	1178
10	178303	223	OK	34	1182	515	179715	16448	OK	No	115	1182

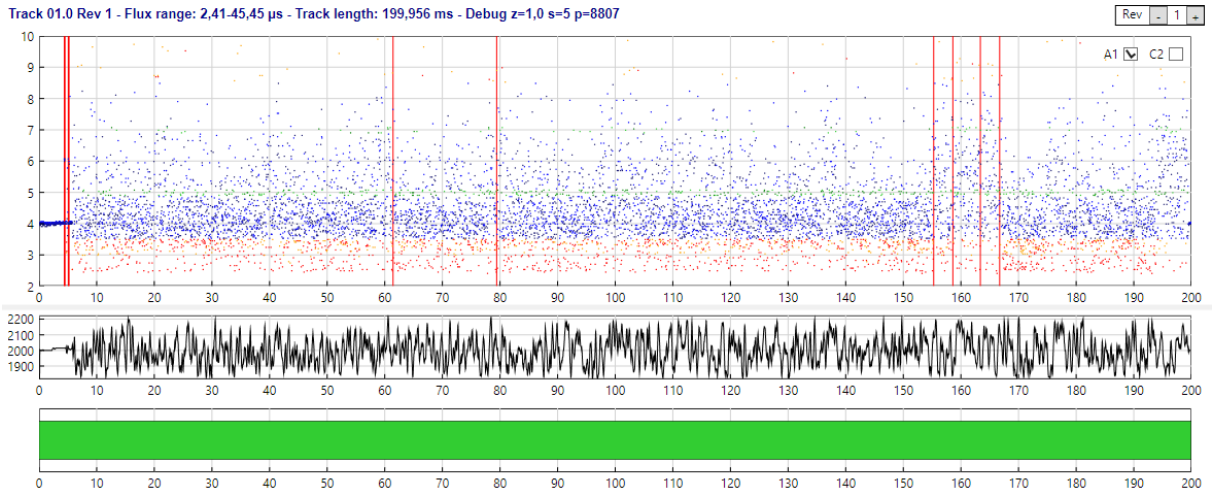
This is the Rob Northen protection from 1988



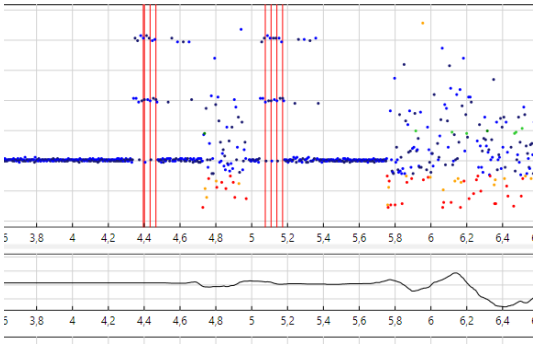
5.16 Power Drift

This game is the only one I know that uses the Fuzzy Track protection.

The protection is located at the beginning of track 2 on Disk 2/3



The track starts with some MFM data and the rest seems unformatted. We can see some sync mark at the beginning of the track. Let's now zoom in this area.



The program probably tests the protection by reading several times the track 01.0.

Here is for example three consecutive reads:

```
00112  2015 003614  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00128  2015 004129  FF FF FF FF FF FF FF C2 A1 A1 A1 A1 00 00 F7 F7  ÿÿÿÿÿÿÿÂ¡¡¡¡..÷÷
00144  2017 004622  F7 F7 00 00 D7 60 31 FF BF F4 21 78 00 00 A1 A1  ÷÷..×`1ÿ¿ô!x..¡¡
00160  2015 005134  A1 A1 00 00 F7 F7 F7 F7 00 00 00 00 00 00 00 00  ¡¡..÷÷÷÷........
00176  2009 005652  00 00 00 00 57 DF F9 F9 FC E1 80 3F E1 F2 A6 40  ....WßùùüáÞáò¦@
00192  2052 006164  6B 38 A4 80 7E 63 E1 DF 18 58 0B 0E 88 00 1C C4  k8¤.~cáß.X.....Ä
00208  1941 006651  00 92 01 28 4F F1 90 38 7E 0F F8 44 40 07 E1 00  ...(Oñ.8~.øD@.á.

00112  2015 003614  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00128  2015 004129  FF FF FF FF FF FF FF E1 A1 A1 A1 A1 00 00 F7 F7  ÿÿÿÿÿÿÿá¡¡¡¡..÷÷
00144  2015 004625  F7 F7 00 00 44 0F 4B 3E 0F 04 84 41 8F FF F0 A1  ÷÷..D.K>...A.ÿð¡
00160  2015 005105  A1 A1 A1 00 00 F7 F7 F7 F7 00 00 00 00 00 00 00  ¡¡¡..÷÷÷÷.......
00176  2009 005623  00 00 00 00 00 53 DF F9 F8 02 1C 31 80 10 09 E2  .....Sßùø..1...â
00192  2038 006133  5F 82 52 28 80 20 E1 04 03 1D 06 23 8D 9F FF 06  _.R(. á....#..ÿ.
00208  2078 006638  80 0B 38 13 34 FF 19 86 0A E0 7F C2 38 00 3E 00  ..8.4ÿ...à.Â8.>.

00112  2015 003617  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00128  2015 004131  00 00 00 00 00 00 00 0A A1 A1 A1 A1 00 00 F7 F7  ........¡¡¡¡..÷÷
00144  2015 004628  F7 F7 00 00 57 E0 34 00 00 4B B7 D0 00 01 A1 A1  ÷÷..Wà4..K·Ð..¡¡
00160  2000 005140  A1 A1 00 00 F7 F7 F7 F7 00 00 00 00 00 00 00 00  ¡¡..÷÷÷÷........
00176  2015 005657  00 00 00 00 57 5F F9 FC 78 02 9C 10 80 4F 03 FC  ....W_ùüx....O.ü
00192  2041 006178  12 09 46 7C 00 67 C0 7C 61 4E 23 8F 20 00 72 CC  ..F|.gÀ|aN#. .rÌ
00208  2130 006663  05 00 95 10 08 C2 C4 80 00 04 98 BE 90 2E BF EE  .....ÂÄ....¾..¿î
```

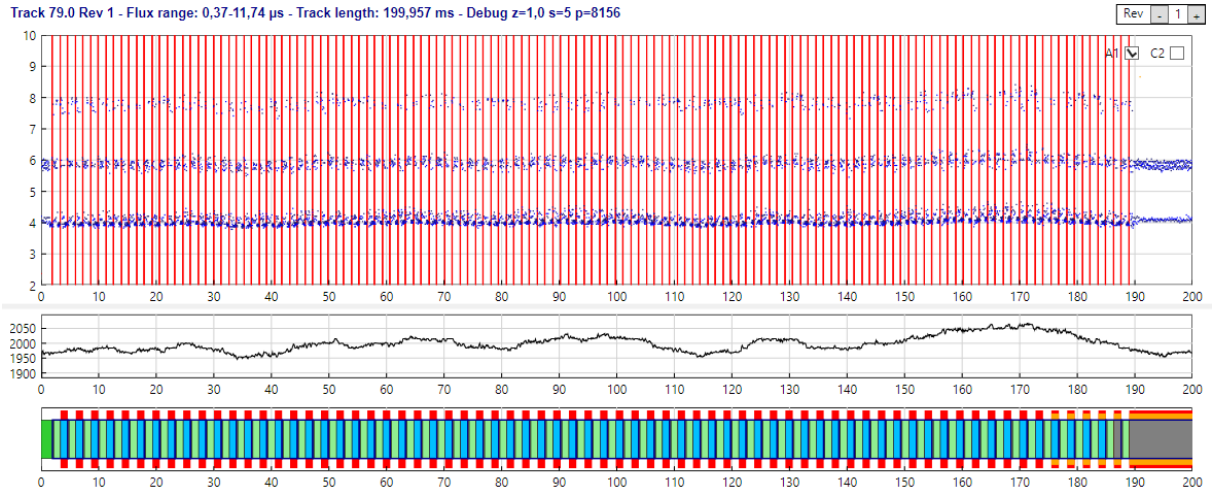
We can see that after the sync marks we have a constant sequence 00 00 F7 F7 F7 F7 00 followed by some fuzzy bytes followed by another sync mark sequence.

Therefore the program probably tests the constant sequence that contains invalid $F7 bytes, and then must test the presence of fuzzy bytes.

This kind of protection (Fuzzy track) is not supported by Pasti format and therefore it is not possible to save a working .stx file for this game.

5.17 Sherman M4

Track 79.0 has 70 sectors. Each sector is declared with a size of 512 bytes and therefore it is obvious that they all overlap (Sector Within Sector) and have CRC errors. Some of them even have fuzzy bytes. The track looks like this

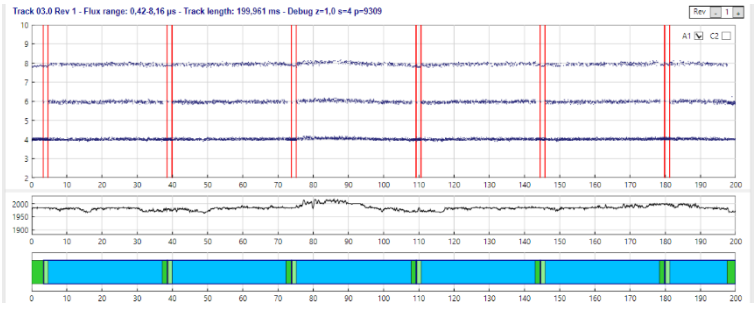


Similar Au Nom de l'Hermine, Fire and Forget …

5.18 Star Glider 2

- Track 00.0 contains just one sector that looks like a boot loader.
- Track 01.0 contains unknown data? (firebird track?)
- Tracks 02.0-78.0
  - 5 sectors of 1024 bytes + 1 sector 512 bytes
  - Sector numbered 245 - 250 ($F5 - FA)
  - All gap data $4E replaced by Invalid bytes $F7
  - Invalid ID Field

```
245   3353 222  OK |   34 1175 | 1027   4760 32576  OK  No |   43  1175
246  38562 221  OK |   34 1172 | 1027  39965 32615  OK  No |   45  1172
247  73813 222  OK |   34 1176 | 1027  75221 32804  OK  No |   41  1176
248 109251 221  OK |   34 1171 | 1027 110652 32608  OK  No |   44  1171
249 144494 222  OK |   34 1175 | 1027 145901 32704  OK  No |   44  1175
250 179844 223  OK |   34 1183 |  515 181260 16406  OK  No |   73  1183
```



Now let's detail the first ID field:

- Track num = $F7
- Side num = $F7
- Sector num = 245 ($F5)
- Size = $F7.

You have to play some trick to bypass the $F7 track number, the side is not used. For the size if only the last two bits are kept we end up with 3 (1024 bytes).

For the last ID of the track the size = $F6 that result in 2 (512 bytes).

```
00096  1984 003075  DF DF DF DF DF DE A1 A1 A1 FE F7 F7 F5 F7 02 72  ßßßßßÞ¡¡¡þ÷÷õ÷.r
00112  1979 003576  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7  ++++++++++++++++
00128  1984 004084  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7  ++++++++++++++++
00144  1984 004592  F7 F7 A1 A1 A1 FB 6A 70 33 D1 B8 C2 43 03 FC 67  ++¡¡¡ûjp3Ñ¸ÂC.üg
00160  1969 005089  C0 00 12 61 C6 00 23 E4 01 01 81 01 9E 00 60 40  À..aÆ.#ä......`@
00176  1992 005597  81 C2 00 03 00 40 20 60 61 8C 48 72 23 80 10 10  .Â...@ `a.Hr#...
00192  1984 006101  18 00 80 1F 80 84 13 90 00 00 41 E0 64 60 31 80          A`d`1
```

```
05680  2000 179480  00 00 00 00 00 00 00 00 01 A1 A1 A1 FE F7 F7 FA  .........¡¡¡þ÷÷ú
05696  2000 179972  F6 02 6D F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7  ö.m+++++++++++++++
05712  2000 180484  F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7  ++++++++++++++++
05728  1997 180996  F7 F7 F7 F7 F7 A1 A1 A1 FB 76 74 FD E1 B0 BF 02  +++++¡¡¡ûvtý á°¿.
05744  1984 181509  10 EA EB 7E 13 EA EB 7E 12 EA 22 D9 FD 58 4C CD  .êë~.êë~.ê"ÙýXLÍ
05760  1984 182018  EE 7A 43 7E 6F 6C F9 5C 4C DA DB 04 2D 28 4C C9  îzC~olù\LÚÛ.-(LÉ
05776  1984 182528  FA 76 6E D5 F3 EA 02 D9 96 42 5A FD F9 60 E8 87  úvnÕóê.Ù.BZýù`è.
```

Of course you can see that all the inter/intra gap bytes are $F7

5.19 Theme Park Mystery (Image Works)

Theme Park Mystery from Image Works uses the following protection mechanisms:

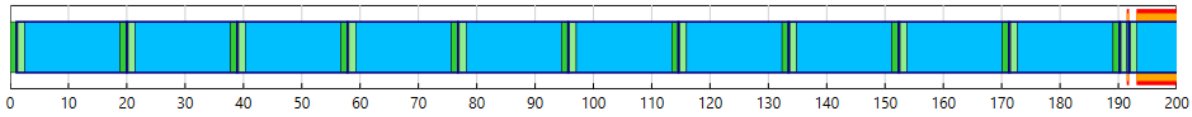
- Number Of Sectors: **12** sectors per track on all tracks!
- Fuzzy Sectors on all tracks
- Sectors with bad *Data* *Field*s on all tracks
- Sector Within Sector on all tracks
- Data Over Index on all tracks
- No Flux reversals area

We are going to describe mainly the Sector within Sector over NFA protection.

Here is a dump of the end of track 1 (sector 11):

```
05920  1984 188224  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
05936  2000 188735  FF FF FF FF FF FF FF FF FF FE 00 20 42 42 42 42  ÿÿÿÿÿÿÿÿÿþ. BBBB
05952  1989 189245  42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  BBBBBBBBBBBBBBBB
05968  1992 189757  42 42 42 43 FF FF FF FF FF FF FF FC A1 A1 A1 FE  BBBCÿÿÿÿÿÿÿü¡¡¡þ
05984  2000 190264  00 00 0B 02 25 A4 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  ....%¤NNNNNNNNNN
06000  2000 190774  4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 00 00 00 00  NNNNNNNNNNNN....
06016  2000 191289  00 00 A1 A1 A1 FB 00 00 00 00 00 00 00 00 A1 A1  ..¡¡¡û........¡¡
06032  2000 191797  A1 FE 00 00 0C 02 BC 33 4E 4E 4E 4E 4E 4E 4E 4E  ¡þ....¼3NNNNNNNN
06048  2000 192309  D9 23 76 C5 E6 D3 31 B2 4E 4E 4E 4E 4E 4E 4E 4E  Ù#vÅæÓ1²NNNNNNNN
06064  1988 192822  FF FF FF FF FF FE A1 A1 A1 FB 00 00 00 00 00 00  ÿÿÿÿÿþ¡¡¡û......
06080  1984 193333  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
06096  1984 193843  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
06112  1989 194333  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
06128  1989 194845  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
06144  1989 195357  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

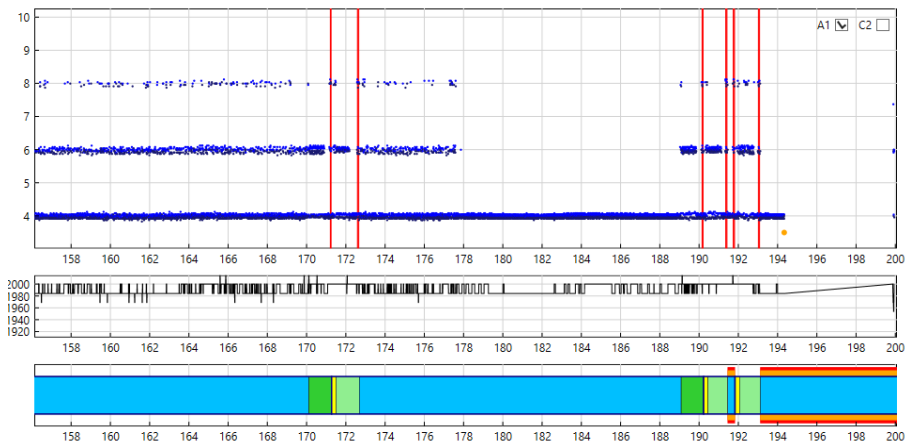
Inside the data block of the sector 11 (0x0B) we have a sync sequence of 3 $A1 followed by an IDAM followed by the *ID Field* for sector 12. Then we have a GAP3 followed by a sync sequence followed by a DAM and the *Data Field*.



Here we clearly see that:

- Sector 10 is normal.
- Sector 11 has a DATA Field wrapping at the beginning of the track (DOI) is read with a CRC error (CRC) and fuzzy bits, ant it contains sector 12 (SWS).
- Sector 12 has DATA field starting inside sector 11 and wraps at the beginning of the track it is also read with a CRC error and fuzzy bits.

We have also a no flux reversal area at the end of the track. After zooming at the end of the track we see the no flux reversal area (NFA):

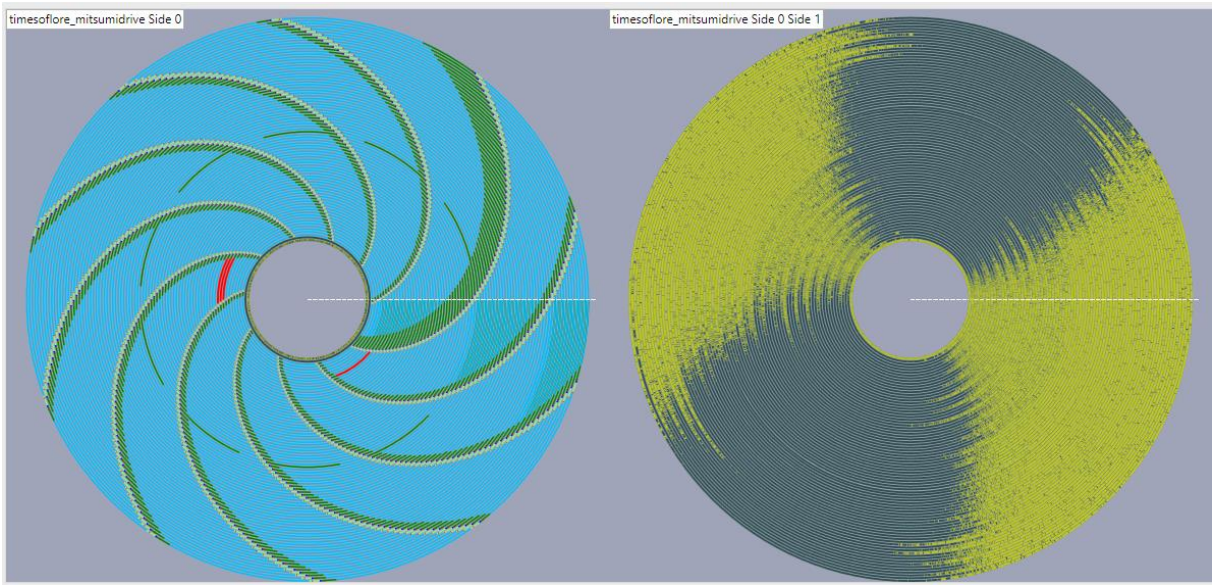


This NFA is located inside the two overlapping sectors 11 and 12 (SWS).

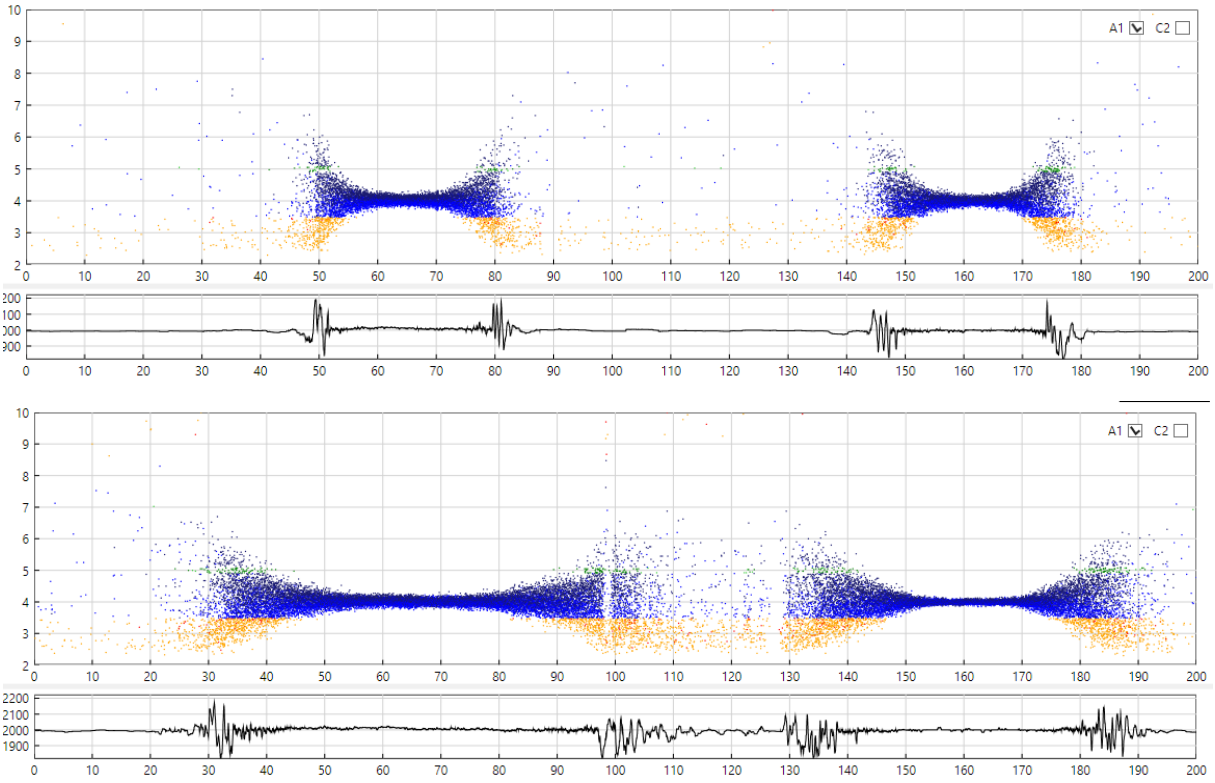
Refer to Kick off 2.

5.20 Time of lore

Most tracks are shifted tracks. What is interesting is that the shifting of the track is proportional to the track number. We have the following pattern



Side 1 of the disk contains some strange flux transitions



But probably not used?

5.21 Turrican

Turrican contains a lot of interesting protection mechanisms. You should refer to information provided by Markus Fritze on Turrican protection and the Atari Forum

We are going to look at the following protections:

- Non-standard sector size 1 * 512 + 5 * 1024 (total of 5632 bytes)
- No Flux reversal Area
- Sector within Sector - with cell bit shifting allowing to read clock bits
- Fuzzy sector with CRC error
- Data Over Index

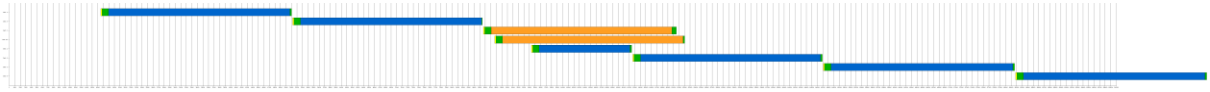
The read track provides the following layout:

```
********************************************************************************
Track Layout Information: 6290 Bytes - length=199.98 ms
ID Good/Bad=4/1 - Data Good/Bad=1/4 - Sync Good/Bad =10/42
********************************************************************************

GAP1 1 bytes length=63.00 us
-----------+------------------+-----------+--------------------------+------------
GAP2       |ID                |GAP3       |DATA                      |GAP4
Bt   Lgt   |Sct Pos    Lgt CRC|Bt  Lgt  BS|Bt   Lgt   CRC TMV BRD Clk |Bt  Lgt   BS
-----------+------------------+-----------+--------------------------+------------
517  16488 |3   16551  222 OK |38  1177  1|1027 32677 BAD 0   0   3.98|13  415    0
6    172   |6   51217  224 OK |37  1185  0|1027 32703 OK  0   0   3.98|4   127    0
7    223   |0   85682  217 BAD|38  1183  0|1027 32503 BAD 0   1   3.96|8   252    0
863  27242 |2   147081 222 OK |37  1175  0|1027 32835 BAD 0   0   4.00|10  318    0
6    191   |5   181825 223 OK |37  1178  0|525  16753 BAD 0   0   3.99|0   0      0
-----------+------------------+-----------+--------------------------+------------
```

We can see that the FD uses several 1024 bytes sector and that the last sector is truncated indicating Data over Index.

The no flux area is located inside the sector 0 but we will see that this sector 0 in fact contains sector 16 and sector 1.



Here we can see clearly that sector 0 contains sector 16 and sector 1 (Sector within sector) and that sector 16 fully contains sector 1 of 512 bytes (sector within sector within sector). We can also see that sector 0 and sector 16 reads with fuzzy bits (orange bar on the plot). The reason is that sector 0 and sector 16 contains the no flux reversal area.

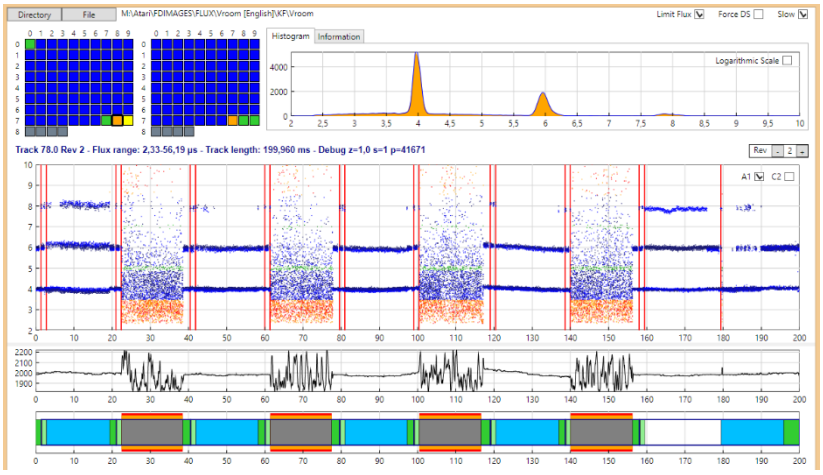
An interesting mechanism is used: Sector 0 start with the following bytes:

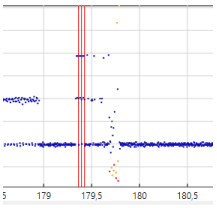
```
Detail buffer content for sector 0 with 1027 bytes
= DATA ID=0 1027 bytes @87082 us length=32637.85 CRC BAD CLK=3.97 TMV=0 BRD=1 DOI=0
  *** Fuzzy Sector *** starting at byte position 217
  0000 87082  4000  fb 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
                    00 7f ff ff ff ff ff ff ff ff ff ff ff ff ff ff  .□..............
  0010 87596  3968  00 a1 a1 a1 fe 07 00 10 03 bb 21 4e 4e 4e 4e 4e  ..........!NNNNN
                    ff 0a 0a 0a 00 f8 7f e7 fc 00 4e 10 90 90 90 90  ......□...N.....
  0020 88103  3968  4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e  NNNNNNNNNNNNNNNN
                    90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
  0030 88614  3968  4e ff ff ff ff ff ff ff ff ff ff ff fe 14 14 14  N...............
                    90 00 00 00 00 00 00 00 00 00 00 00 00 a1 a1 a1  ................
  0040 89124  4000  00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
                    fb 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0050 89629  3968  ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

In green we can see inside data block the presence of 3 sync character followed by the ID block of sector 16 (sector within sector) however if we look further we do not see directly the sync mark for the data block. Instead we see the presence of 3 bytes with value $14 followed by a byte $00. If we turn the “clock” flag of the ***KFAnalyze*** program it also print the clock value of the decoded byte. Here we can see that in fact the $A1 sync bytes are in fact in the “clock” bytes. The sync mark detector will take care of shifting by a half cell to correctly read the data of sector 16. This result in reading the “data” bytes for sector 0 and reading the “clock” bytes for sector 16. This allow to check the presence of the NFA (no clock, no data).

5.22 Vroom

- Track 00.0 + 77.0 + 78.1 + 79.1 seems to be normal tracks.
- Tracks 01.0-76.0 + 00.1-76.1 are Data Tracks. They seems to us the same format as described in Maupiti Island using a $07 escape character (Lankhor ST format?).
- Track 78.0 is extremely strange. It contains 9 sectors alternatively formatted and unformatted. At the end of the track we find a data segment followed by another data segment. In other word a data segment not preceded by an ID segment. There is no way to read this kind of segment with a ***read sector*** command however it is possible to test the presence with a read track command. But here is the interesting protection: This sector contains the normal $A1 $A1$ A1 $FB header that is used to sync correctly the WD1772 but after we see some transitions that violates the normal MFM codding and this results in fuzzy bytes. So if we do several consecutive read track and look at the beginning of this segment we have:

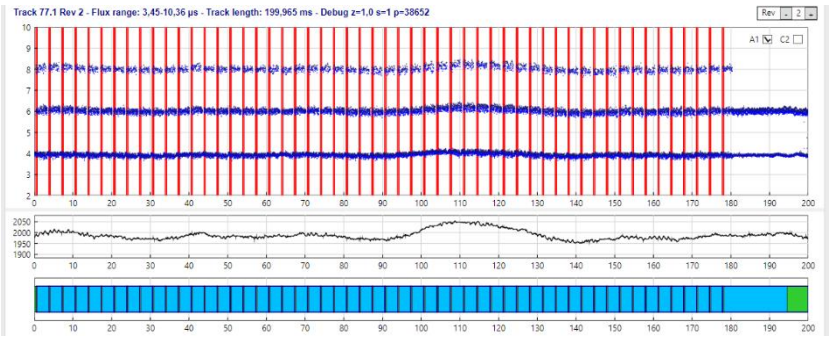




```
rev 1 we have A1 A1 A1 FB 00 4E 80 00 E7 7F FB 7F F7 BF 7F FF FF FF FF…
rev 2 we have A1 A1 A1 FB 00 4E 80 00 E7 7F FB 7F E5 BF 7F FF FF FF FF…
rev 3 we have A1 A1 A1 FB 00 4E 80 00 E7 7F FB FF F5 00 80 00 00 00 00…
Etc.
```

  So we can see that at the beginning of this pseudo sector we have track fuzzy bytes. Note that this kind of protection cannot be imaged with the Past format. See also Power Drift.
- Track 79.0 is also strange it contains 2 $A1 sync mark followed by a constant set of MFM flux at 4µs decoded as a continuous sequence of $00 bytes.
- Track 77.1 contains 54 overlapping sectors. But with an extremely strange pattern I have never seen before. The data field follow the ID field immediately **without any gap bytes**! Normally a read sector command should not work as no time is given to the WD1772 to "react" to the detection of the correct ID. Note that the sort data segment contains in French the message: *"Hello. Vroom protection track 77 / 79 Christian and Costas. Thanks."*

```
00000  1970 000029  00 00 00 00 00 00 00 00 00 00 00 02 A1 A1 A1 FE  ..............¡¡¡þ
00016  2000 000525  4D 01 01 02 AA 45 A1 A1 A1 FB 01 42 6F 6E 6A 6F  M....ªE¡¡¡û.Bonjo
00032  1984 001035  75 72 20 21 20 56 52 4F 4F 4D 20 70 72 6F 74 65  ur ! VROOM prote
00048  2000 001550  63 74 69 6F 6E 20 70 69 73 74 65 20 37 37 20 2F  ction piste 77 /
00064  2000 002065  20 37 39 20 43 68 72 69 73 74 69 61 6E 20 65 74   79 Christian et
00080  2004 002576  20 43 6F 73 74 61 73 2E 20 4D 65 72 63 69 2E 4E   Costas. Merci.N
00096  2000 003087  4E 4E 4E 4E 4E 4E 4E 4E 00 00 00 00 00 00 00 00  NNNNNNNNN.......
00112  1984 003601  00 00 00 00 00 A1 A1 A1 FE 4D 01 02 02 FF 16 A1  .....¡¡¡þM...ÿ.¡
00128  2015 004111  A1 A1 FB 02 42 6F 6E 6A 6F 75 72 20 21 20 56 52  ¡¡û.Bonjour ! VR
00144  2000 004626  4F 4F 4D 20 70 72 6F 74 65 63 74 69 6F 6E 20 70  OOM protection p
00160  2006 005138  69 73 74 65 20 37 37 20 2F 20 37 39 20 43 68 72  iste 77 / 79 Chr
00176  2000 005653  69 73 74 69 61 6E 20 65 74 20 43 6F 73 74 61 73  istian et Costas
00192  2010 006169  2E 20 4D 65 72 63 69 2E 4E 4E 4E 4E 4E 4E 4E 4E  . Merci.NNNNNNNN
00208  2000 006684  4E 4E 00 00 00 00 00 00 00 00 00 00 00 00 A1 A1  NN............¡¡
00224  1981 007195  A1 FE 4D 01 03 02 CC 27 A1 A1 A1 FB 03 42 6F 6E  ¡þM...Ì'¡¡¡û.Bon
```

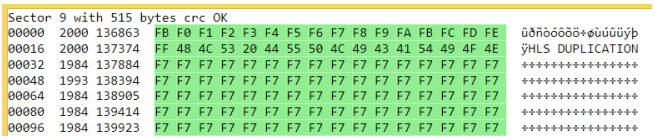


5.23 Wizball, Ocean

This is supposed to be old Copylock from 1987?

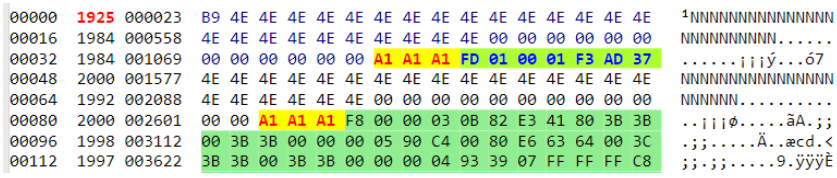
- Track 79.0
  - Data over index sector 246
  - Invalid sector number 245, 246,247 ($F5, $F6, $F7)
  - Sector 9 inside sector 247 (SWS)
  - Invalid character $F7 in GAP
  - Hidden data in track

Most of the protections can be seen in the above section of the read track. We see that inside sector 247 ($F7) we have a short data field that contains only $F7 bytes. This is followed by sector 9 ID field, followed by $F7 invalid characters in a GAP. Then we have the data segment with F0F1F2…FF484CC200… and the sector continue filled with $00 bytes. However if we read this sector 9 with a read sector command we find F0F1F2…FF484C followed by 49434154494F4E (DUPLICATION).



5.24 Z-out

- Track 00.0 is a normal track.
- Tracks 00.1-79.1 use special format:
  - 6 Sectors by track: 1024, 1024, 1024, 1024, 512, 1024
  - Invalid ID Field (IDF)



The normal $FE IDAM is replaced by $FD. As you know the WF1772 ignore the last bit of the AM therefore $FD=$FC. Normally $FC is an IAM (Index address mark) located after a sequence of three $C2 sync mark used in DOS diskettes but not used in Atari diskettes. So this is quite strange but indeed an undocumented feature of the WD1772 is to consider the $FD IAM to be equivalent to an $FE IDAM. To summarize it is possible to read sector with an ID starting with the $FD IAM. The next strange thing is the length field of the ID field. Normally it should be in the range 0-3 but here the value is $F3. In fact another not documented feature of the WD1772 is that it only looks for the last two bits in the length field and ignore the other bits. Therefore $F3 is interpreted as $03 corresponding to a size of 1024. The sector with a length of 512 bytes is described with `A1 A1 A1 FD 01 00 05 A2 2B 27 4E 4E 4E 4E 4E 4E`

Here again the length $A2 is interpreted as $02 corresponding to a size of 512 bytes.

Chapter 6. References

6.1 Documents / Articles

- Article on protection "copy me I want to travel" from Claus Brod the expert who wrote the book Scheibenkleiste covering all sort of interesting details about floppy disks, hard disks, RAM disks, CD-ROMs and other mass storage devices for the Atari (Claus web site).
- Probing the FDC: *Learn the Secrets of your Floppy* - By David Small
- Atari Protected Disk Image Format & Atari Preservation Project
- Floppy disk format How can I copy my copy-protected Atari software
- An interview with Rob Northen
- Dungeon Master Copy Protection
- Disk Backup Programs: Do they really work
- Teac & Citizen Micro Floppy Disk Drive Specification
- Floppy from HP
- How to HD install Pacland (MFM format) using WHDLoad
- Commodore C1581-handler
- S100-Manuals - Disks and Disk Drives
- Wipe Swap File
- SpinRight Technical note
- An interview with Rob Northen

6.2 Forums Threads

- Way how SW testing copy protection
- Analysis of submitted games
- List of difficult to copy disks
- Floppy Disk Copy Protection
- Copy Protection details
- Looking for Rob Northen originals
- Rob Northern Code Found
- Weak Bits, Bit-rate var., data under index: Copy Protection
- Questions Regarding STT Images
- Protected disk images project & CAPS
- Ideas about ST floppy image make program for PC
- PASTI Project
- Copy II ST
- Looking for AntiBitos 1.4 by illegal
- Most memorable Hack/crack
- Protected Disk Image Project Seeking Beta Tester
- Ideas about ST floppy image make program for PC
- Looking for DMA file under interrupt
- Mega STE Specifics
- Copy Protected Disks at AtariAge
- Gcopy DIM file
- ST Protection routines
- Putting a second internal floppy drive in the STF
- RamDisk and ATARI-ST Disk IO
- X-out original protected
- Copy Protected Disk – Turrican NFA Protection (IFW / Mr. Vince)
- List of Difficult to Copy Disks

6.3 Related Patents

You may want to look at the following patents that describe some protection mechanisms:

- Copy Protection for computer Disc 4,849,836
- Computer Program protection method 4,462,078
- Hardware key-on-disc for copy protecting magnetic storage data 4,577,289
- Copy protecting system for software protection 4,584,641
- Techniques for preventing unauthorized copying of information recorded on a recording medium and a protected recording medium 4,734,796
- Copy protection disc format controller 5,432,647
- Data Input Circuit with Digital Phase Lock Loop

6.4 Web Sites

- Atari ST FD (Hardware view)
- Atari ST FD (Software view)
- Atari FD Protection/Preservation
- Atari ST Copy Protections (Markus Fritze)
- Protections sur Atari ST/Amiga
- PASTI Project
- Software Preservation Society
- KryoFlux Products & Services Limited
- C64 Preservation Project (Commodore)
- Atari Disk Image FAQ
- Tim Mann's TRS-80 Pages
- The .ADF (Amiga Disk File) format FAQ
- Introduction to Magnetic Recording
- Funny presentation about perpendicular magnetic recording !!!
- Individual Computer Support
- The Central Point Option Board
- SpinRite's Defect Detection Magnetodynamics
- The Gentle art of Protection
- The XCOMP/2 home page
- LIBDSK library for accessing discs and disc image files
- WinUAE Amiga Emulator

6.5 FDC & Related Information

- Atari ST – FD/HD Programming – Jean Louis-Guérin
- WD1772 Floppy Disk Formatter/Controller - WD Corporation – JLG edition
- Western Digital Corporation 5.25" WD1770/1772 Floppy Disk Controller/Formatter
- 8272 SINGLE/DOUBLE DENSITY FLOPPY DISK CONTROLLER
- Intel 82077AA FDC Datasheet
- Commodore C1581 - WD1770 FLOPPY DISK CONTROLLER
- PC87310 (SuperI/OTM) Dual UART with Floppy Disk Controller and Parallel Port
- Hard Disk Data Encoding / Decoding.
- Cyclic Redundancy Check, CRC16-CCITT, The Great CRC Mystery Terry Ritter

6.6 Game References

- Arkanoid: ST Protection M.Fritze
- Au nom de l'hermine: List of difficult to copy disks
- Barbarian Pasti/STX File Format Ways how SW testing copy protection ST Protection M.Fritze
- Colorado: List of difficult to copy disks ST Protection M.Fritze
- Eco: List of difficult to copy disks (more more)
- Falcon: Analysis of submitted games

- Ghost Buster: List of difficult to copy disks
- Golden Axe: List of difficult to copy disks
- Indiana jones and the last crusade: ST Protection M.Fritze
- Jupiter Masterdrive: List of difficult to copy disks (more)
- Maupiti Island: List of difficult to copy disks ST Protection M.Fritze
- Midi Maze: ST Protection M.Fritze
- Power Drift: List of difficult to copy disks Power Drift and Pasti & patch (more) Way how SW testing copy protection
- Spy Vs Spy: ST Protection M.Fritze
- Start Glider 2: List of difficult to copy disks
- Time of lore: SCP Disk images Way how SW testing copy protection
- Turrican: ST Protection M.Fritze
- Vroom: Pasti images that should but don't work
- Wizball: List of difficult to copy disks
- Z-out: List of difficult to copy disks (more)

Chapter 7. Document history

- V1.3a – November 14, 2014 – Fixed hyperlink not working
- V1.3 - November 12, 2014. The categories of protection is now reduced to two (removed fuzzy and physical – not used – categories). The fuzzy bit detailed description is now moved as a section of the Useful information chapter and fuzzy sector/track added to data category. NFA moved to timing category. Names for protections changed to more intuitive names. Added section of write splices and sync marks. Added Game reference section. Added **many** new games analysis. Added Chapter on preserving floppy disks.
- V1.2 June 2014 – Lots of information added, regrouping of related protections, new examples, etc. Most significant is Unformatted Tracks, No Sector Data Track, Partially unformatted track, Fuzzy Data Track, No Flux Area on Disk, Unformatted Diskette / Track / Sector …
- V1.0 - November 2011. Added information on low level format, particularly about the write splice. Added description about ***KFPanzer*** and ***KFAnalyze***. Now the analysis of games uses the output from ***KFAnalyze*** and especially the nice plots. Added the Short/Long Track and No Flux reversal Area protections. Remove documentation of ***Analyze*** program. Added more analysis of games (Turrican and others). New information about games protection based on new ***KFPanzer*** capabilities. Added more links to new sites. Added reference to the new **KryoFlux** board and related - After 5 years of development I consider the document mature enough to go to version 1.0
- V0.9 – September 2010. Clean-up text based on feedback. Modified documentation to reflect the usage of the new **Panzer** (Protection ANalyZER) program. Added ID Fuzzy Bits, Invalid Data in Gap, and Non Standard DAM Protection. Added a *section on Preservation* for each of the protections. Added description for Barbarian, Operation Neptune Game. Work with Gothmog (Christophe Fontanel) on getting more accurate information on Dungeon Master fuzzy bits protection
- V0.8 – October 2007. Added taxonomy for the different protection categories. Rewrote of large portion of the explanations about fuzzy bits. Added 5 new protections: Invalid ID Field, Non Standard IDAM, Sector over Index pulse, Missing Track and Sector within Sector. Added description for several games (Theme Park Mystery, Computer Hits Volume 2, Kick Off 2, Colorado). Better documented Intra-sector Bit Variation with reference to Colorado. For the first time lots of diskettes (over 50) have been tested and references for them have been entered in the document. And again lots of clean-up
- V0.7 – January 2007. Several modifications based on feedback from Ijor and Obo. Added a new section on weak bits based on US patent and a section on Invalid character during format. Plus lots of miscellaneous cleanup.
- V0.6 – December 2006. Modifications based on feedback from Ijor, I have added one section about Double Density diskette format, the Invalid sector number protection, and the intra-sector variable bit rate protection – December 2006
- V0.5 – December 2006. Incorporated feedback from Gothmog about the DM protection patent, added a section with several US patent about protection, modified the section on fuzzy bits, modified the fuzzy bit detection in WD1772 DPLL
- V0.4 - November 2006. Complete documentation cleanup and links verification.
- V0.3 - October 2006. Major Revision: Merged several related sector protections, modified extensively the description of several protections, added section on example of protections, added analyze program short presentation, added DPLL presentation, and added new protections: PAT and NAT.
- V0.2 - June 2006. Minor correction based on feedback.
- V0.1 - May 2006. Initial writing.